The Development of Processing Schedules when Successful Service Completion is Unobservable and/or Uncertain

Emma Punton

Submitted for the degree of Doctor of Philosophy

The University of Edinburgh

2007

Abstract

This thesis examines service systems where there is some uncertainty over the successful completion of jobs within the system. Such problems have not been widely studied, with much research assuming that jobs are willing to await service indefinitely and that completion can be immediately observed during processing.

However, there are instances where this may not be the case. These include the allocation of firepower in the military context, where 'jobs' are enemy targets who may move out of range during 'service'. Another example for which our models are relevant is a situation in which some service schedule is to be produced in advance, where the exact time of completion is not known.

In these situations, allocating a large amount of processing to one job may increase its chances of being successfully completed but will impose a burden on other jobs awaiting service. Thus, any schedule should include some such 'trade-off'.

We examine this trade-off in three broad types of problem. In the first type, a discount factor is applied to future rewards and fixed-time schedules developed. For the second type, fixed-time schedules are developed for a single class queueing system, subject to job arrivals and losses during service. In the third type, a multiple class queueing system is studied, allocating a group of servers between job classes.

For each problem, stochastic dynamic programming optimality equations are formulated. However, dynamic programming suffers from the well documented 'curse of dimensionality' with problems becoming increasingly more difficult to solve as the number of states increases. Thus, the focus of this thesis is the development of heuristics as alternatives. Such heuristics are developed using results from queueing theory and their effectiveness is discussed. Characterisations of optimal schedules are also presented.

I declare that this thesis is my own work and that the work contained in it has not been submitted for any other degree or professional qualification.

Two papers arising from this work have been published (Glazebrook and Punton (2005)) or accepted for publication (Glazebrook and Punton (2007)) before the submission of this thesis. This was done with the approval of the supervisors and the papers are contained in the Appendices.

Signed

Date 28th-Nov-2008

Acknowledgements

I would firstly like to thank Prof. Kevin Glazebrook for his encouragement, support and supervision from my initial application to study at The University of Edinburgh through to the completion of this thesis. I am extremely grateful for his continued help.

I would also like to thank Dr. John Glen for his assistance, comments and useful suggestions in the final stages of this thesis.

Thanks to my family for their love and support. Throughout my studies, as with everything I have done, they have been very proud and had total belief in my abilities.

Thanks to my friends in Edinburgh, they all made my time here much more enjoyable and I have fond memories of our dinners together. A particular mention must go to my fiancé, Leo. He has helped me see there is a world of possibilities, has always used the perfect words of encouragement and without him my life would be far more dull.

Lastly, I am very grateful to the EPSRC for providing the funding which has made it possible for me to undertake this research.

Contents

Chapter 1

Introduction

We examine decision problems in which a server (or a group of servers) is concerned with the scheduling of jobs while unable to immediately observe whether service has been successful. For example, consider a doctor in a hospital casualty department faced with a number of patients to examine. During an initial examination of each patient, the doctor cannot know exactly how long it will take to treat the patient nor can they necessarily see whether the treatment given will ultimately be successful and the patient cured. If the doctor were to concentrate all their effort on treating one patient then this patient's successful recovery may be more likely but many more patients would experience delays in receiving treatment, during which their conditions may deteriorate. Thus, there is an obvious scheduling dilemma for the doctor where they need to choose how long to spend with each patient and which ones should be given priority. One response to this dilemma could be to decide some fixed time to spend on an initial examination of each patient so that they can be stabilised until further treatment can be offered.

Alternately, consider a military situation where a 'Blue' agent is under attack from a number of 'Red' agents. The Blue agent seeks to successfully 'serve' (eliminate) the Red agents while the Red agents seek to avoid service. A Blue agent may only be able to target one Red agent at any time, and may not be able to observe immediately whether any firepower allocated has destroyed the enemy

agent. This could apply to long range targets, where satellite images are required to visualise the target area and so there is a natural delay before receiving the image. There may also be uncertainty over how long the Red agents will be available for 'service' as they move out of range after their objective is complete. Concentrating a large amount of firepower on an individual Red agent may make it more likely to be eliminated but would leave other Red agents free to carry out their attack, unhindered. Thus the Blue agent may adopt a strategy whereby they allocate firepower to each Red agent for a fixed (but limited) length of time, after which another Red agent is targeted.

Both of the above examples are among those cited by Gaver et al. (2006) as instances of service systems in which the successful completion of service cannot immediately be observed, and which can be addressed using techniques from queueing theory. However, such situations have largely gone unstudied.

In this thesis, we seek to address this gap in research, examining systems which can be used to model these kinds of situations where there is uncertainty over the successful completion of service. We develop policies which dynamically allocate service to each job (e.g. patient/agent) in a way which depends on the number of jobs (or patients/agents) present in the system.

1.1 Markov Decision Processes

The scheduling problems described fall into the category of Markov decision processes. In Markov decision processes, the state of the process at the next observation depends only on the current state of the process, and the decision taken at the current time, and is independent of the past behaviour of the process. Thus, for the example of the doctor the number of patients present the next time the system is observed depends only on how many patients there are now and how long the doctor gives to the patient next served. In particular, it is independent of how many patients there were say an

hour or a day earlier, or at any other times previously.

In a Markov decision process, the state of the process is observed at each decision epoch and an action applied. The state of the process then changes according to the dynamics determined by the action applied, and a reward is earned. This change in state is termed a state transition.

At each time, $t=0,1,2,...$, the state of the system can be classified into one of a number of possible states where the set of possible states is written as I. For each state $i \in I$, a decision must be made on what action to take, with the set of possible actions available dependent on the current state of the process. The set of possible actions in state i is written as $A(i)$. If the action $a \in A$ is chosen in state i then an immediate reward $R(i,a)$ is earned and the system will occupy state j at the next decision epoch with probability $p_{ij}(a)$, where $\sum_{j \in I} p_{ij}(a) = 1$. The aim is to maximise some measure of the total expected reward earned over the lifetime of the system.

We now formally define the Markovian property which is exhibited by Markov decision processes. According to the Markovian property, the state the process enters at the time $t+1$ depends only on the current state, i, of the system and the decision a taken at the current time, t. This state transition is independent of the states occupied at previous times s, $0<s<t$. Therefore, the past history of a process need not be considered when making a decision, only the current state of the system. This Markovian structure greatly simplifies the problem and thus the framework provided by Markov decision processes is very useful for systems in which a sequence of decisions are to be made. Not only does it limit the amount of information required in order to make a decision, and so simplify the analysis required, but it also leads to the development of *stationary policies* and the possible application of some theoretical results which further simplify analysis.

If the next state transition is independent of past state transitions, then it is possible to apply policies which allocate service such that the same action will be

taken at each entry to a particular state, irrespective of the time at which the state is entered, or the past behaviour of the system before entering this state. These are called stationary policies. Stationary policies form a subset of all possible policies and are simply of the form 'whenever in state n, apply action a'. One method for obtaining an optimal stationary policy is Dynamic Programming. This will be discussed later in this chapter and utilised throughout the thesis.

There is a large body of literature on the general theory of Markov decision processes. See, for example Tijms (1994), Puterman (1994) or White (1993). One area in the literature which exploits the characteristics of Markov decision processes is *renewal theory*. Renewal theory is useful when a state can be defined (generally the empty state) where the evolution of the process is probabilistically identical at each entry into this state. This means that each time the system enters this state, it can be considered to 'renew' and thus the long-run evolution of the process can be considered as the disjoint union of a number of independent periods between two consecutive renewal points. This will be further explained and applied within the thesis.

1.2 Multi-Armed Bandit Problem

One class of Markov decision processes is that of multi-armed bandits. The multi-armed bandit problem is so called after the problem faced by a gambler using a fruit machine with a number of arms about whose reward characteristics (s)he is uncertain. At each play, the gambler can pull one arm and can either succeed or fail in winning a prize. The problem is how to sequentially pull the arms in order to maximise some measure of the expected total reward earned. This problem has applications in the contexts of the sequential design of experiments (see, for example, Glazebrook (1980)), oil exploration (Benkherouf et al. (1991)) or the scheduling of jobs (Gittins (1979)).

In the multi-armed bandit problem, a set of M projects are competing for processing effort. At each decision epoch one project (or arm), say m, is chosen to be processed and stochastically evolves to earn a discounted (expected) reward $\beta^t r_m\{X_m(t)\}$, where $X_m(t)$ is the state of the Markov chain modelling the evolution of project m at time t, r_m is a bounded reward function of the project's state and $\beta \in (0,1)$ is the discount factor applied. All projects not chosen for processing remain in their current state at time t and no reward is earned from them. Only the projects being processed evolve and earn a reward.

The aim is to find a policy for processing the projects to maximise the total expected reward earned over an infinite horizon (or until all projects are completed). The multi-armed bandit is a classical problem. However, it has proved extremely difficult to solve.

1.3 Dynamic Programming

Dynamic programming (DP) is a systematic approach to obtaining an optimal sequence of decisions for a dynamically changing system. It was introduced by Bellman (1954) and can be applied to Markov decision processes where a sequence of decisions are required, including the multi-armed bandit problem. DP provides a framework for formulating a problem in terms of value functions which can then be solved recursively over the lifetime of the system using backwards induction. By backwards induction, the solution is developed starting at the end of the planning horizon, with the final decision, and working backwards to the first decision.

The DP formulation is often expressed as a recursion and a boundary condition. Let $V_n(i)$ denote the maximum expected reward for an n-stage problem (meaning there are n decision epochs until the end of the planning horizon) that starts in some

state i. The recursive expression is

$$V_n(i) = \max_a [R(i,a) + \beta \sum_j P_{ij}(a) V_{n-1}(j)], \tag{1.3.1}$$

with some known boundary condition, say $V_0(i) = 0, i \in I$, where $R(i,a)$, $P_{ij}(a)$ and β are as defined previously. The aim is to maximise some measure of the total expected reward. The expression is evaluated for successive decision epochs, n, so that at each n the action yielding the highest total reward is chosen and the resulting values of $V_n(i)$ can be used in the next recursion for $n+1$.

DP is a powerful technique as it is guaranteed to provide the optimal solution (Bellman (1957)) in a range of problems and is much more efficient than calculating all possible solutions to a problem (usually called complete enumeration). It is applicable to Markov decision processes where sequential decisions are to be made. For general discussions of dynamic programming see Bellman (1957), Ross (1983) or Hillier and Lieberman (2005).

However, the main drawback of DP is that as the size of the state space in a problem increases, the number of calculations required for each step of the recursion also increases. Problems of a reasonable size can quickly become computationally cumbersome, particularly so for cases in which the state space is multi-dimensional. This is often referred to as the 'curse of dimensionality'. This provides motivation in many important problem contexts for seeking more efficient methods for developing optimal policies if they exist or, when they do not exist or cannot easily be found, for developing heuristics which can be more easily computed and yield solutions which are close to optimal.

1.4 Gittins Index

Dynamic programming remained the primary method of obtaining optimal solutions for the multi-armed bandit problem until Gittins and Jones (1974) elucidated an optimal policy in the form of an index policy. Their proposed policy could be thought of as a forwards induction policy. This was in contrast to the backwards induction of DP described above. The policy proposed by Gittins and Jones (1974) utilised a calibrating index (later called the Gittins Index) which was calculated for each project in its current state. The optimal policy mandated that the project with greatest index should be selected for processing.

This was a very powerful result as the calibrating index for a project only required information about the individual project itself. Hence, an m-project multi-armed bandit problem was effectively reduced to m single-armed bandit problems.

Thus the policy is relatively simple to implement: calculate the index for each project, then process the project with highest index. The indices for all projects not processed remain unchanged so at the next decision epoch only the processed project need have its index recalculated. The indices for all the projects are now compared and again the project with highest index allocated processing. However, while the policy itself was simple, the proof of optimality was very difficult to follow and so the result was not widely recognised immediately. This led to alternative proofs of optimality being presented. The original proof was based on an interchange argument and simpler interchange arguments were later presented by Weiss (1988), Varaiya et al. (1985) and Tsitsiklis (1994). Other proofs included a DP approach by Whittle (1980), and also an argument by Weber (1992) which avoided much of the detailed calculations of other approaches. Bertsimas and Niño Mora (1996) provided a proof via an achievable region approach and Katehakis and Veinott (1987) discussed the approximate optimality of an index related rule, leading to a short proof of the optimality of the Gittins Index. Gittins (1979) and Gittins and Jones (1979) provided

further discussion of the index and its calculation, with Gittins (1989) giving a more detailed description of the topic.

A feature of the multi-armed bandit problem is that processing of projects can be interrupted and they will remain in the same state as they await further processing. It is by virtue of this property that the Gittins index policy is optimal. Therefore, choosing to process one project at any given decision epoch does not prevent another project from being processed at a future decision epoch.

However, this freedom to interrupt processing of projects at zero cost has led to policies obtained using Gittins indices which often involved an unacceptable amount of switching between projects. This can yield unwanted switching costs and so Banks and Sundaram (1994) examined the extent to which the Gittins index policy remained optimal when there are costs placed on switching between jobs, while Benkherouf et al. (1993) aimed to remedy the issue by developing a class of so-called single-visit policies.

1.5 Restless Bandits

A generalisation of the class of multi-armed bandits was introduced by Whittle (1988). In these restless bandits, projects that are not being processed are also able to stochastically evolve according to so-called passive dynamics. This yields a more difficult problem than the multi-armed bandit and appropriate indices do not always exist for the restless bandit problem. When indices do exist, they may not yield an optimal solution. Whittle (1988) developed an index based heuristic for problems which are indexable (i.e., for which appropriate indices can be defined), providing conditions for which the index may exist. Whittle (1988) also showed that the multi-armed bandit is a special case of a restless bandit and that in this case, when state transitions are static in the absence of service, Whittle's index reduces to the Gittins index.

Papadimitrou and Tsitsiklis (1999) showed that restless bandits cannot be solved in polynomial space (they are PSPACE hard) and so the problem of determining optimal policies is likely to be intractable. Thus research has concentrated on the development of heuristics in such problems. For example, Niño Mora (2001) demonstrated conditions under which a class of restless bandit problems are indexable. He also proposed the use of an algorithm to determine whether problems belong to this class. Glazebrook et al. (2002) developed an approach to policy evaluation of a general class of indexable discounted restless bandits. Weber and Weiss (1990) and Weber and Weiss (1991) looked at the properties of an index policy and conjectured that it might be asymptotically optimal under certain conditions. Dacre et al. (1999) also sought optimal solutions by using an approach which characterised a space where solutions may be achievable.

The job scheduling problem in Chapter 2 can be viewed as a restless bandit. It is always indexable in the case of a fixed number of jobs to be processed. In this thesis, we will use the ideas developed by Gittins and Whittle.

Other extensions to the multi-armed bandit problem have included arm-acquiring bandits (Whittle (1981)) and generalized bandits (Nash (1980)).

1.6 Queueing Systems with Impatient Customers

A large body of research exists on queueing systems in which new jobs enter a system over time. These systems can be interpreted in the context of call centres, where the jobs are customers contacting the call centre, or in the context of manufacturing operations requiring the production of manufacturing schedules. In the call centre context, customers may be impatient and hang up if kept waiting for too long, or in the manufacturing context, projects may have due dates. Problems with known due dates (see for example, Doytchinov et al. (2001) or Jiang et al. (1996)) do not require the same analysis as problems involving impatience in which the time that the job

will leave the system is unknown.

Research which has investigated impatience in the above contexts includes that of Garnett et al. (2002), Bassamboo et al. (2005), Harrison and Zeevi (2004) and Glazebrook et al. (2004). Garnett et al. (2002) and Bassamboo et al. (2005) considered the impact impatience may have on the design of call centres while Glazebrook et al. (2004) cited the context of perishable goods which may become unusable after some unknown deadline. The latter paper discussed a restless bandit model for impatient customers and a model of a multi-class M/M/1 queueing system with losses. In this case longer queues were favoured for service. However, it is often found that in analyses of multiclass queueing systems members of a lower priority class must wait longer for service and that these waits are less predictable. This was a feature already identified in multi-class scheduling models so an attempt to address this was made by Ansell et al. (1999) who developed a programming approach to minimising the expected queue lengths, so limiting this effect. Most of the research on the control of queueing systems has concentrated on systems in which the server is able to immediately observe successful completion. Such research does not cover the contexts described by Gaver et al. (2006) and which are outlined at the beginning of this chapter, where successful completion is not observable.

So far, only Gaver et al. (2006) have developed models in which the successful completion of service is also not immediately observable. They considered static policies where some constant time is allocated to each job, that time to be chosen so that the number of losses from the system is minimised. One of the outstanding issues discussed in their paper is that of the development of heuristics which dynamically allocate processing. This issue is one we address in this thesis.

1.7 Thesis Outline

The general aim of this thesis is to look at situations in which a server, or group of servers, has to choose how to allocate processing when faced with a number of jobs to serve. This could be either a collection of jobs which are to be served (a clearing system), or one in which jobs could arrive over time. The server allocates processing but is unable to observe in real time whether the processing has achieved a successful completion of service. Thus, in all systems the server chooses to allocate a fixed amount of service to each job. The service allocated to each job is in general dependent on the number of jobs currently present in the system.

In this situation, there is a trade-off between allocating a large amount of service to the current job to ensure that it is successfully completed and delaying waiting jobs, thus risking their loss from the system through impatience. This trade-off would lead us to conjecture that more service should be allocated to a job when fewer jobs are awaiting service, or when smaller rewards are earned by later jobs. The server's aim is to develop a schedule which allocates processing for each job in order to maximise a measure of total expected reward.

We consider a number of models based around this theme. In all the models considered, optimal solutions could in principle be obtained using DP methods. However, in recognition of the 'curse of dimensionality' related to DP, we seek alternatives to this. Our aim is to determine (where possible) the characteristics of optimal schedules and to develop some strongly performing heuristics, evaluating their performance.

This thesis is structured as follows. In Chapter 2 a number of models are examined for which the penalty imposed for postponing awaiting jobs is represented by a simple reward structure where future rewards are discounted. A server aims to allocate a fixed time to each job in a way which maximises the total expected discounted reward.

Firstly, a general case is considered where a collection of non-identical jobs is to

be processed by a single server. Theoretical results are developed using ideas from work on multi-armed and restless bandits, and two heuristics are proposed. Secondly, a model is considered for which a server has a *batch* of jobs, where all the jobs in the batch have identical characteristics. Results are presented for this model describing how fixed processing times should be chosen for each job. Thirdly, the above batch model is extended to a multiple class model, so that each job to be processed can be classified into one of a number of job classes according to its characteristics. Results are again presented, describing both the fixed-time chosen and the ordering of job classes. Lastly, a model is considered incorporating the arrival into the system of new jobs over time.

In Chapters 3 and 4 we concentrate on more complex queueing systems. In both chapters, a penalty is imposed for postponing awaiting jobs so that they may be lost from the system unprocessed. The time at which these jobs may be lost from the system is unknown to the decision maker. In Chapter 3 a single server seeks to give fixed times to jobs of a single class in order to maximise the average rate of successfully served jobs. A dynamic programming solution is presented and two heuristics are constructed which utilise the idea of policy improvement. The performance of the heuristics is discussed.

The model examined in Chapter 4 is a multiclass queueing system such that jobs arrive at a central group of servers. These servers are to be allocated among the job classes in order to maximise the reward rate earned over all job classes. In both Chapters 3 and 4, heuristics are developed and evaluated. Numerical examples are presented and discussed in all cases. Finally, in Chapter 5 the main findings of this research are summarised and possible directions for future research discussed.

Chapter 2

Discounted Fixed-Time Schedules

2.1 Introduction

In this chapter we examine systems in which a server is presented with a collection of jobs and, unable to observe their successful completion during processing, chooses to allocate a fixed time to each job. After this time the job will be halted and another job processed. Thus the server seeks some way to decide how long to allocate to each job and also the order in which to process the jobs.

When deciding the time allocated to a job, the server must consider that while a large block of processing may make the successful completion of the current job more likely, it will mean a greater delay for all awaiting jobs. We investigate this trade-off between the current and future jobs by discounting all future rewards, so that the later a job is completed, the smaller the reward earned from it.

One example of an application of this model is the situation where processing schedules are being planned in advance of a project. Clearly, at the planning stage the server will not know whether any time they give will be sufficient to successfully complete the job (or element of the project), but they may have an idea of the probability that the job will be completed.

Such a system can be modelled as a restless bandit, a class of problems introduced by Whittle (1988). For restless bandit problems, a server chooses one of a number of projects to process at each decision epoch and this project changes state

with some known probability structure, earning a reward. The remaining projects are also able to change state, however no reward is earned. Whittle (1988) showed that restless bandit problems are intractable in general and also developed a heuristic for allocating service.

Restless bandits were developed as an extension to the classic multi-armed bandit problem for which jobs can only change state while being processed. Thus, the multi-armed bandit problem is a special case of the restless bandit problem, where the state transitions under no service allocation are simply to remain in the same state. Multi-armed bandit problems have been widely studied and Gittins and Jones (1974) developed an optimal policy which utilised a relatively simple calibrating index for each project. The heuristic developed by Whittle (1988) was related to Gittins' index and so also took the form of an index policy. These indices will be used extensively throughout this chapter.

The feature which makes our problem different to multi-armed bandit problems is that we restrict processing so that any single job is only eligible to be processed once. If a job is given some processing then halted in favour of another job, it can not be given any further processing. This is in contrast to the problems Gittins and Jones (1974) studied where jobs could be halted and resumed later. In fact, one of the limitations of their index policy is that in some cases it led to an unacceptable amount of switching between jobs.

Benkherouf et al. (1993) aimed to limit this by introducing switching costs when changing the job to be processed. These costs should discourage unnecessary switching between jobs. They also introduced a class of policies which only allowed jobs to be processed once, namely single-visit policies. Methods for the development of optimal single-visit policies were presented, with these policies expressed in terms of Gittins indices.

In this chapter, we consider a situation in which a server develops a fixed-time schedule determining both the order for processing the jobs and also the fixed-time allocated to each one. The jobs are processed in order according to the schedule so that once the processing time allocated to a job has expired, the job is discarded and the next job processed for its allotted time. The discarded job could have been successfully completed or could still require processing but is no longer available for further processing. Thus only single visit policies are considered.

While successful completion is not observable, the probability of successful completion is known. Therefore, the objective is to produce a schedule which maximises the total expected reward earned from all jobs. For this type of problem, it would seem reasonable to expect jobs yielding larger rewards and/or which can be completed more quickly to be scheduled earlier so that a smaller discount is applied to the rewards earned from jobs later on in the schedule. We shall investigate whether this is the case.

The remainder of this chapter is comprised of six sections. In Section 2.2, we first present the general case of a collection of non-identical jobs awaiting service and then introduce some theoretical results used to characterise optimal schedules. We then model our system as a restless bandit in Section 2.3, proposing two heuristics based on the Whittle index and illustrating their performance. In Section 2.4, we examine the case in which all jobs are identical and present more theoretical results describing the structure of optimal schedules. This is then extended, in Section 2.5, to the situation where service is to be given to a group of jobs consisting of a number of job classes with jobs in each class being identically distributed. In Section 2.6 we present some theoretical results applying to the processing of jobs when the system is subject to further jobs arriving during processing. And finally, in Section 2.7, we summarise the main findings of the chapter.

2.2 General Fixed-Time Schedule Problem

In this section we examine the general case where a single server is presented with a set of distinct jobs. We first define the problem and describe a dynamic programming approach which gives an optimal solution. We then present some results to aid in the development of alternative approaches for developing an optimal schedule, using a calibrating index.

2.2.1 Problem Set-up

The problem is defined as a server presented with a set $J=\{1,2,3,...,N\}$ of distinct jobs with the following properties:

- Each job $j \in J$ has processing requirement C_j, a random variable taking values in the positive integers, subject to a maximum possible value of T_j. This processing requirement is assumed to be independent for each job. Thus we write

$$p_j(s) = P\{C_j = s\}, 1 \leq s \leq T_j, j \in J,$$

  as our probability that job j is completed in exactly s time units. We assume that $p_j(s)$ is known and

$$\sum_{s=1}^{T_j} p_j(s) = 1.$$

- Each job j also has an associated positive reward r_j which is subject to a discount rate $\beta \in (0,1)$. If job j is completed at time t the reward $\beta^t r_j$ is earned. Thus job j earns expected reward

$$R_j(t) = r_j \sum_{s=1}^{t} \beta^s p_j(s), 1 \leq t \leq T_j, j \in J,$$

  when processed from time 0 up to time t.

- There are no restrictions on the order in which jobs are processed, thus we write $(\pi, \mathbf{t})$ as our fixed-time schedule, where π is a permutation of J and $\mathbf{t}$ is an

N-vector, whose j^{th} component is an integer in the range $[1, T_{\pi_j}]$. Our schedule tells us that at time 0 we process job π_1 until time t_{π_1}, then job π_2 from time t_{π_1} until $t_{\pi_1}+t_{\pi_2}$, and so on. Thus, for a set of three jobs with $\pi = (2, 1, 3)$ and $\mathbf{t} = (5, 3, 7)$ we would first process job 2 for 5 units, then job 1 for 3 units from time 5 until time 8, and finally job 3 for 7 units from time 8 until time 15. We assume that switching between jobs is instantaneous and incurs no costs.

The total expected reward associated with schedule $(\pi, \mathbf{t})$ is given by

$$V(\pi, \mathbf{t}) = \sum_{k=1}^{N} \beta^{\sum_{j=0}^{k-1} t_{\pi_j}} R_{\pi_k}(t_{\pi_k}).$$

Our aim is to find the schedule $(\pi, \mathbf{t})$, representing the ordering and set of processing times, which maximises $V(\pi, \mathbf{t})$.

This problem set up allows us to explore some of the properties of the schedules produced, examining the trade-off between allocating processing to those jobs which yield large rewards or jobs yielding smaller rewards but which are more likely to be successfully processed in a shorter time.

2.2.2 Dynamic Programming Approach

In the search for an optimal schedule, it is possible to calculate the total expected reward for all possible combinations of job orderings and allocated processing times (complete enumeration) and hence choose the solution yielding greatest total expected reward as the optimal solution. However, this would require a considerable number of calculations for any problem of reasonable size. For a problem with N jobs and each job j requiring a maximum processing time T_j, total enumeration would require $N!(\Pi_{j=1}^{n} T_j)$ calculations.

We therefore present an alternative to complete enumeration, namely dynamic programming (DP). The DP method is a systematic method for solving a problem recursively and can greatly reduce the number of calculations required. Moreover,

DP is guaranteed to yield the optimal schedule (Ross, 1983).

The DP recursion for this problem begins with the case where there is only one job present and then develops optimal solutions for increasingly large sets of jobs, using the solutions obtained at the previous step. We now present our DP recursion for this problem and describe in detail how it is used. The DP recursion equation is:

$$\begin{aligned} V(S) &= \max_{j \in S, 1 \leq t \leq T_j} [R_j(t) + \beta^t V(S \backslash \{j\})], S \in 2^J \\ V(j) &= R_j(T_j), \forall j \in J, \end{aligned} \tag{2.2.1}$$

where $R_j(t)$ is the expected reward earned when job j is processed from time 0 up to time t, as defined in the previous section, and $V(S)$ is the total expected reward gained from an optimal fixed-time schedule for subset $S \subseteq J$. This total expected reward, $V(S)$, is comprised of $R_j(t)$ for the current job j and the total expected reward earned from the optimal schedule for the subset $S \backslash \{j\}$. The decision is optimised for all choices of time for a given job and for all choices of current job, j, from within S.

When optimising the time, we use the convention that if there are several t maximisers in (2.2.1) for some j then we always take the largest as our allocated processing time.

In order to further clarify the use of the DP algorithm for our problem, we now show how it is applied in more detail.

We begin with all sets of a single job, $j \in J$, for which we plainly have

$$V(j) = R_j(T_j). \tag{2.2.2}$$

We then calculate the optimal schedules for all pairs of jobs, $(i, j) \in J$, using (2.2.2) and

$$V(i,j) = \max \left\{ \max_{1 \leq t \leq T_i} [R_i(t) + \beta^t V(j)], \max_{1 \leq t \leq T_j} [R_j(t) + \beta^t V(i)] \right\}. \tag{2.2.3}$$

We repeat this process for all triples, $(i, j, k) \in J$, using the solutions for all pairs from

(2.2.3) and

$$V(i,j,k)=\max\left\{\max_{1\le t\le T_i}[R_i(t)+\beta^t V(j,k)],\max_{1\le t\le T_j}[R_j(t)+\beta^t V(i,k)],\max_{1\le t\le T_k}[R_k(t)+\beta^t V(i,j)]\right\}, \tag{2.2.4}$$

and so on with all sets of four, then five, etc., jobs, until we have our optimal solution for the full collection of the set of J jobs.

The DP approach described above, while indeed greatly reducing the number of calculations required for complete enumeration, is still exponential in N. From (2.2.4) we see that for a set of 3 jobs we already have three alternative choices of current job which each need to be maximised over t. As the number of jobs increases, the number of alternative choices of initial job also increases and, thus, DP becomes burdensome for problems for which N is large.

This limitation of the DP approach, therefore, leads to our interest in finding alternative methods for producing optimal schedules, or in the development of strongly performing heuristics. We seek methods which will yield solutions close to optimal but which are more easily calculated than DP.

We now present a result relating to optimal fixed-time schedules. This result will be used in the development of our proposed heuristics and throughout the remainder of this chapter.

Lemma 2.2.1. *If $S_1 \supset S_2$ then $V(S_1) > V(S_2)$.*

Proof. Let us construct a schedule for S_1 which chooses to order the jobs according to the optimal fixed-time schedule obtained for the set S_2, followed by any ordering of the jobs in $S_1 \backslash S_2$. Times are also allocated to jobs in S_2 according to the optimal schedule for S_2 and some choice of processing times is given to the remaining jobs. This schedule gives a total expected reward of $V(S_2)$ plus some positive contribution from the jobs in $S_1 \backslash S_2$ so is clearly greater than $V(S_2)$. Thus the optimal fixed-time schedule for S_1 must be at least as good as this and the result follows. □

The above proof explains our assertion that all jobs should be allocated at least one unit of processing. Should a schedule allocate no processing time to a job j, a higher expected reward can be earned simply by moving this job to the end of the schedule and allocating some processing to it. The total expected reward earned from all the other jobs would remain unchanged but now job j would contribute some (heavily discounted) reward. Thus an optimal fixed-time schedule would never allocate 0 processing to any job.

The idea that all jobs will be allocated some processing and that an optimal schedule for a set of jobs must yield a greater reward than the optimal schedule for a subset of these jobs will be utilised repeatedly throughout this chapter and when developing our proposed heuristics.

2.2.3 The Gittins Index

We begin our search for alternative methods for developing optimal schedules with an examination of the recurrence equation (2.2.1). The issue with DP and direct use of (2.2.1) arises from the requirement to simultaneously decide both the job to be processed and the time allocated to it. However, if we simplify the decision by imagining we have already chosen a job and concentrate on only optimising the time allocated, we note that we can derive an index from recurrence equation (2.2.1). This index will be central to future analyses of the characteristics of optimal schedules and to the development of our proposed heuristics.

The idea is to fix the job under consideration and thus determine some properties of the time which yields highest total expected reward for this choice of job. Our derivation is as follows:

For a given S and choice $j \in S$, consider

$$V(S) = \max_{1 \leq t \leq T_j} [R_j(t) + \beta^t V(S \backslash \{j\})],$$

and suppose the time τ achieves the maximum. Then, by the maximality of τ, it

must be true that

$$R_j(\tau)+\beta^{\tau}V(S\backslash\{j\}) \geq R_j(\tau+u)+\beta^{\tau+u}V(S\backslash\{j\}), 1 \leq u \leq T_j-\tau.$$

Rearranging this gives

$$\begin{aligned} V(S\backslash\{j\})(\beta^{\tau}-\beta^{\tau+u}) &\geq R_j(\tau+u) - R_j(\tau) \\ V(S\backslash\{j\}) &\geq \max_{1\leq u\leq \tau-T_j}\left\{\frac{\beta^{-\tau}\left[R_j(\tau+u)-R_j(\tau)\right]}{(1-\beta^u)}\right\}. \end{aligned} \tag{2.2.5}$$

We have therefore shown that the time yielding the greatest total expected reward is such that the index on the right hand side of (2.2.5) is less than, or equal to, the total expected reward earned from the other jobs to be processed. This index is independent of the other jobs in S, and can be calculated based only on information about the job j under consideration. Thus the index could facilitate our search for optimal schedules and we make the following definition:

Definition 2.2.1. The **Gittins index** $G_j : \{0, 1, ..., T_j\} \rightarrow \mathbb{R}^+$ is given by

$$G_j(s) = \max_{1\leq t\leq T_j-s}\{G_j(s,t)\}, 0\leq s\leq T_j-1,$$

where

$$G_j(s,t) = \frac{\beta^{-s}\{R_j(s+t)-R_j(s)\}}{1-\beta^t}, 1\leq t\leq T_j-s, 0\leq s\leq T_j-1.$$

We set $G_j(T_j)=0$ and call $G_j(0)$ the *initial index.*

This index can be quantified for all times, for each job in S, and so may allow us to determine the maximising value of t, rather than using the DP method.

Before describing some properties of optimal schedules in relation to the Gittins Index, recall that in our derivation of the Gittins Index, the maximisation of (2.2.1) for a particular job j was shown to be equivalent to the search for a time t such that $G_j(t) \leq V(S\backslash\{j\})$. We can, however, place a stronger condition on t.

Lemma 2.2.2. *The largest time t maximising equation (2.2.1) for a job j is such that*

$$G_j(t) < V(S\backslash\{j\}).$$

Proof. For a given job $j \in S$, let t be the time maximising equation (2.2.1) and let s be a time achieving $G_j(t)$. Then

$$\begin{aligned} G_j(0,t+s)(1-\beta^{t+s}) &= G_j(0,t)(1-\beta^t)+G_j(t,s)(\beta^t-\beta^{t+s}) \\ &= G_j(0,t)(1-\beta^t)+G_j(t)(\beta^t-\beta^{t+s}). \end{aligned}$$

Now suppose that $G_j(t) \geq V(S\backslash\{j\})$. Therefore s satisfies

$$\begin{aligned} R_j(t+s)+\beta^{t+s}V(S\backslash\{j\}) &= G_j(0,t+s)(1-\beta^{t+s})+\beta^{t+s}V(S\backslash\{j\}) \\ &= G_j(0,t)(1-\beta^t)+G_j(t,s)(\beta^t-\beta^{t+s})+\beta^{t+s}V(S\backslash\{j\}) \\ &\geq G_j(0,t)(1-\beta^t)+\beta^t V(S\backslash\{j\}) \\ &= R_j(t)+\beta^t V(S\backslash\{j\}), \end{aligned}$$

which contradicts t as the largest maximiser of (2.2.1). Hence $G_j(t) < V(S\backslash\{j\})$. □

In our search for optimal schedules, we use the Gittins Index to define the following two sets:

Definition 2.2.2. The set of *maximal times* for job j is written $Max(j)$ and is a subset of $\{1, 2, ..., T_j\}$ defined by

$$Max(j) = \{t; G_j(0,t) > G_j(0,s), t < s \leq T_j\}.$$

(i.e. the set of times such that no larger time provides a higher value as an option for the initial Gittins index.)

The set of *minimal times* for job j is written $Min(j)$ and is a subset of $\{1, 2, ..., T_j\}$ defined by

$$Min(j) = \{t; G_j(t) < G_j(s), 0 \leq s \leq t-1\}.$$

(i.e. the set of times such that the Gittins index for this time is smaller than the Gittins indices for all earlier times.)

We write t_j^* for the largest t-value achieving the initial index, i.e.

$$t_j^* = \max\{t; G_j(0,t) = G_j(0)\}.$$

The following proposition demonstrates the relationship between $Min(j)$ and $Max(j)$, both of which will be of interest later in the chapter.

Proposition 2.2.3. *For all j we have*

$$\{t_j^*, T_j\} \subseteq Min(j) \subseteq Max(j).$$

Further, $t \in Max(j) \implies t_j^* \leq t \leq T_j$.

Proof. From the above definitions it is clear that $T_j \in Min(j) \cap Max(j)$ and also that $t_j^* \in Max(j)$. It is also obvious that t_j^* is the smallest member of $Max(j)$ (by the definition for t_j^*). Further, as part of the central theory on multi-armed bandits, explained in Gittins (1989), t_j^* can be characterised as

$$t_j^* = min\left\{t; G_j(t) < G_j(0)\right\}. \tag{2.2.6}$$

Thus by the definition of $Min(j)$, $t_j^* \in Min(j)$.

We now need to show that $t \in Min(j) \implies t \in Max(j)$. Suppose that the set of $t \in Min(j) \backslash \{t_j^*, T_j\}$ is non-empty. Write $\eta(t)$ for the number of minimal times for job j which are no larger than t. Order these times as

$$s_1 < s_2 < .... < s_{\eta(t)} = t,$$

and write $s_0 = 0$. A slight extension of the result at (2.2.6) shows that time $s_{m+1} - s_m$

achieves the index $G_j(s_m), 0 \leq m \leq \eta(t)-1$. Thus we have

$$\begin{aligned} G_j(0,t)(1-\beta^t) &= \sum_{m=0}^{\eta(t)-1} G_j(s_m, s_{m+1}-s_m)(\beta^{s_m}-\beta^{s_{m+1}}) \\ &= \sum_{m=0}^{\eta(t)-1} G_j(s_m)(\beta^{s_m}-\beta^{s_{m+1}}) \\ &> G_j(t)(1-\beta^t), \end{aligned}$$

where the inequality follows from the minimality of t. Thus, from this and the definition of the Gittins index, we have

$$G_j(0,t) > G_j(t) \geq G_j(t,u), 1 \leq u \leq T_j - t.$$

It now follows that

$$\begin{aligned} G_j(0,t+u)(1-\beta^{t+u}) &= G_j(0,t)(1-\beta^t)+G_j(t,u)(\beta^t-\beta^{t+u}) \\ &< G_j(0,t)(1-\beta^t)+G_j(0,t)(\beta^t-\beta^{t+u}) \\ &= G(0,t)(1-\beta^{t+u}), \end{aligned}$$

and so

$$G_j(0,t) > G_j(0,t+u), 1 \leq u \leq T_j - t.$$

Hence $t \in Max(j)$ and $Min(j) \subseteq Max(j)$. □

We have shown that t_j^* and T_j belong to the special sets $Min(j)$ and $Max(j)$. The importance of these sets was noted when examining results from numerical examples. It was observed that DP only selected times which belonged to $Max(j)$, and later that only belonged to $Min(j)$. This observation led to the development of the following theorem.

Theorem 2.2.4. *Optimal fixed-time schedules contain only minimal times.*

Proof. We only need to show that any time, τ, which achieves the maximum in the right hand side of DP equation (2.2.1), for some $j \in S$, must be in $Min(j)$. We do this by assuming that such a τ is not in $Min(j)$ and obtaining a contradiction.

From Lemma 2.2.1 we first note that, for this particular j and S,

$$V(S) = R_j(\tau) + \beta^\tau V(S\backslash\{j\}) = G_j(0,\tau)(1-\beta^\tau)+\beta^\tau V(S\backslash\{j\}) > V(S\backslash\{j\}).$$

By rearrangement and simplification of this inequality we infer that

$$G_j(0,\tau) > V(S\backslash\{j\}).$$

If τ is not in $Min(j)$ then there must exist some $u, 0 \leq u \leq \tau-1$ such that

$$G_j(u,\tau-u) \leq G_j(u) \leq G_j(\tau).$$

As

$$G_j(0,\tau)(1-\beta^\tau) = G_j(0,u)(1-\beta^u)+G_j(u,\tau-u)(\beta^u-\beta^\tau),$$

then after some rearrangement and from Lemma 2.2.2 we have

$$\begin{aligned} G_j(0,u)(1-\beta^u) &\geq G_j(0,\tau)(1-\beta^\tau)-G_j(\tau)(\beta^u-\beta^\tau) \\ &> G_j(0,\tau)(1-\beta^\tau)-V(S\backslash\{j\})(\beta^u-\beta^\tau), \end{aligned}$$

and thus

$$\begin{aligned} R_j(u)+\beta^u V(S\backslash\{j\}) &= G_j(0,u)(1-\beta^u)+\beta^u V(S\backslash\{j\}) \\ &> G_j(0,\tau)(1-\beta^\tau)+\beta^\tau V(S\backslash\{j\}) \\ &= R_j(\tau)+\beta^\tau V(S\backslash\{j\}). \end{aligned}$$

However this contradicts the assertion of τ as a maximiser so it must be true that $\tau \in Min(j)$. □

Theorem 2.2.4 could certainly aid computations which use the dynamic programming equation, limiting the set of times which need to be considered. This is particularly useful as we also know (from Proposition 2.2.3) that optimal fixed-time schedules contain only maximal times which are much more easily obtained than minimal times, and it is often the case that the set $Max(j)\backslash Min(j)$ is small or

even empty. Thus maximal times alone could allow us to gain some insight into the optimal policy for any given example. We will return to the concept of maximal and minimal times later in the chapter when their application will be further explored.

2.3 Whittle Heuristic

The Gittins Index is modelled on the situation in which a job only changes state while being processed. In such situations processing can be given to a job, paused while the server switches to another job, and then service of the first job resumed at some later time, from the point at which it was left. However, for our model we are restricting processing so that a job is only given one opportunity to be processed. Once allocated processing has been halted the job cannot be resumed, amounting to a change in state for this job in the absence of processing. Thus, in our search for heuristics we now present an alternative formulation of our problem to that presented in the previous section.

In our model, a job is only eligible for processing once, so a job given processing at time t but not at time $t+1$ will not be available for processing at time $t+2$. If we view this as causing a change in state for the job while it is not being processed (at time $t+1$), our problem can now be modelled as a *restless bandit process.* This is a family of decision processes introduced by Whittle (1988) as an extension of the multi-armed bandit problems, for which state transitions occur not only while processing is applied (termed the 'active' phase), but also in the absence of processing (in the 'passive' phase).

Whittle showed that restless bandits are not always indexable but presented conditions for which they are indexable and a method of developing indices when they do exist. This method was based on the idea of offering some subsidy W to jobs which were not being processed. The value of W chosen such that both the active and passive options were equally attractive was Whittle's index. Whittle also indicated

that index policies which always choose to activate a project of highest index may not be optimal for restless bandit processes in general but suggested that they could be close to optimal under certain conditions, and presented examples for which this was the case.

We will now concentrate on a single job in our problem to show how it can be considered as a restless bandit and thus how an index can be developed, following the W-subsidy approach used by Whittle (1988).

For a job with reward r, probability of completion $p(\cdot)$, maximum processing time T and discount factor $\beta \in (0,1)$, our restless bandit formulation is as follows:

- The job has state space $\{0,1,...,T\} \cup \{*\}$ such that state $s \in \{0,1,...,T-1\}$ represents the amount of processing already given to the job, state T indicates that no further rewards can be earned from the job, and state $\{*\}$ indicates that the job is not available for processing;

- The server chooses to apply one of two actions $\{a, b\}$ to the job at each decision epoch. Action a is active and indicates processing is to be applied. Action a is admissible in states $\{0,1,...,T\}$ but not in state $\{*\}$. Action b is passive and indicates processing is not to be applied. Action b is admissible in all states.

- State transitions are defined as: If action a is applied to the job in state $s \in \{0,1,...,T-1\}$ at some time t then the job's state at time $t+1$ will be $s+1$ and the expected (discounted) reward $\beta r p(s+1)$ will be earned. If action a is applied to the job in state T then the job does not change state and no reward is earned. If action b is applied, one of three state transitions is possible: $0 \to 0, * \to *$, or $s \to *, 1 \leq s \leq T$. All transitions under the passive action earn a discounted reward $W \in \mathbb{R}$ (the 'subsidy for passivity'). The transition $s \to *$ is what leads to the *restlessness* of the job and characterises our fixed-time scheduling problem. It is this restlessness which makes the problem so difficult.

We now have a Markov Decision Problem in which at each time $t \in \mathbb{N}$ an action for the job is chosen from the admissible set of actions in order to maximise the total expected reward earned over an infinite horizon. Stochastic DP theory (eg Puterman (1994)) guarantees the existence of an optimal policy which is stationary. Thus we can restrict our search for policies to those which are stationary.

The value function corresponding to the optimal stationary policy also satisfies the optimality equations of dynamic programming and we now develop the DP equations for our problem. We write $V(\cdot, W)$ for the maximum expected reward for the W-subsidy problem from initial state $\cdot$. The DP equations relating to our job are as follows:

$$V(0,W) = max\{\beta rp(1) + \beta V(1,W); W + \beta V(0,W)\} \tag{2.3.1}$$

$$V(s,W) = max\{\beta rp(s+1) + \beta V(s+1,W); W + \beta V(*,W)\}, 1 \leq s \leq T-1$$

$$V(T,W) = max\{\beta V(T,W); W + \beta V(*,W)\}$$

$$V(*,W) = W(1-\beta)^{-1}. \tag{2.3.2}$$

By using (2.3.2) to substitute for $V(*,W)$, and also noticing that $V(0,W)$ appears on both sides of equation (2.3.1), we see that the above equations are equivalent to

$$V(0,W) = max\{\beta rp(1) + \beta V(1,W); W(1-\beta)^{-1}\} \tag{2.3.3}$$

$$V(s,W) = max\{\beta rp(s+1) + \beta V(s+1,W); W(1-\beta)^{-1}\}, 1 \leq s \leq T-1 \tag{2.3.4}$$

$$V(T,W) = max\{0; W(1-\beta)^{-1}\} \tag{2.3.5}$$

$$V(*,W) = W(1-\beta)^{-1}. \tag{2.3.6}$$

Equations (2.3.3) to (2.3.6) have an identical reward for the passive action, namely $W(1-\beta)^{-1}$. This suggests that the W-subsidy problem is equivalent to a retirement problem in which at each state $s \in \{0, 1, ..., T-1\}$ a choice is made between allocating a further unit of processing or retiring and collecting the retirement reward of $W(1-\beta)^{-1}$. Obviously, for state $\{*\}$ we would always choose to retire (we have no

choice) and for state T our choice of action is also clear - the allocation of more processing would be optimal for $W \leq 0$ while retirement would be optimal for $W \geq 0$.

Whittle (1980) studied such retirement problems and described optimal policies via the introduction of a set of states such that on entry into this set, it is optimal to abandon the job rather than to continue processing. We write $\Omega(W)$ for the set of states in which it is optimal to retire in our W-subsidy problem. Following earlier analysis, we know for which values of W it is optimal to retire in states T and $\{*\}$. We now need to determine the conditions under which state s is in $\Omega(W)$. From equation (2.3.4), state s is in $\Omega(W)$ when

$$\beta rp(s+1) + \beta V(s+1, W) \leq W(1-\beta)^{-1}.$$

and so from Whittle (1980) and earlier analysis, we infer that when s is in $\Omega(W)$, it must be true that $G(s) < W(1-\beta)^{-1}$. Thus, the set of states for which retirement is optimal is given by

$$\Omega(W) = \begin{cases} \{*\} & W < 0, \\ \{T, *\} \cup \{s; 0 \leq s \leq T-1 \text{ and } G(s) \leq W(1-\beta)^{-1}\} & W \geq 0. \end{cases} \tag{2.3.7}$$

We can now see that our job, as formulated as a restless bandit, fulfils Whittle's (1988) condition for being indexable as stated below:

Definition 2.3.1. The job is indexable if $\Omega(W)$ is increasing in W. If it is indexable then the Whittle index $W : \{0, 1, ..., T\} \to \mathbb{R}^+$ is defined by

$$W(s) = inf\{W; s \in \Omega(W)\}, 0 \leq s \leq T.$$

It is clear from (2.3.7) that $\Omega(W)$ is increasing in W. The definition states that, to create the Whittle index for each value of s, we choose the smallest value of W which will include s in the passive set i.e., the smallest retirement reward we would need to offer in order to induce passivity.

Theorem 2.3.1. *The job is indexable with Whittle index given by*

$$W(s) = \begin{cases} (1-\beta)G(s), & 0 \leq s \leq T-1, \\ 0 & s = T. \end{cases}$$

The above follows directly from Definition 2.3.1 and by rearrangement of the inequality in (2.3.7).

We now consider the full fixed-time scheduling problem set up at the beginning of this chapter with job set $J=\{1,2,...,N\}$. Each job $j \in J$ is formulated as a restless bandit as described and has Whittle index $W_j(s) = (1-\beta)G_j(s), 0 \leq s \leq T$. The heuristic proposed by Whittle (1988) chooses, at each decision epoch,to process a job with the largest value of Whittle index. As β is constant for all jobs, we only need to compare the values of $G_j(s)$ when choosing which job to process.

Thus, according to Whittle's proposal, we use the following procedure

1. Renumber the jobs such that

$$G_1(0) \geq G_2(0) \geq ... \geq G_N(0);$$

2. Process the jobs in numerical order with job j being allocated processing time $\hat{t}_j$, where

$$\hat{t}_j = min\{t : G_j(t) < G_{j+1}(0)\}, 1 \leq j \leq N-1 \tag{2.3.8}$$

and where $\hat{t}_N = T_N$. This is our **'Whittle index heuristic'**.

Note that from (2.3.8) and Definition 2.2.2 it will follow that the allocated $\hat{t}_j \in Min(j), 1 \leq j \leq N$.

The Whittle heuristic provides us with both an ordering for our jobs and a method for allocating processing times to them. However, while we would expect the resulting schedule to perform well, we cannot guarantee that the schedule is optimal. This leaves us with the opportunity to improve upon the schedule, using the following result.

Theorem 2.3.2. *For a given set of allocated processing times* $\mathbf{t} = \{t_1, t_2, ..., t_N\}$*, the optimal ordering is to number the jobs* $\{1, 2, ..., N\}$ *so that*

$$\frac{R_1(t_1)}{1-\beta^{t_1}} \geq \frac{R_2(t_2)}{1-\beta^{t_2}} \geq ... \geq \frac{R_N(t_N)}{1-\beta^{t_N}},$$

and to use this numerical ordering.

Proof. The proof uses a pairwise interchange argument as discussed in Ross (1983). Suppose we have a permutation, $\pi_{(1)}$ of jobs with processing times given. We then create a new permutation $\pi_{(2)}$ by swapping the ordering of two adjacent jobs i and j where $i, j \in J$. This gives us total expected reward for permutation $\pi_{(1)}$

$$R_1(t_1)+\beta^{t_1}R_2(t_2)+...+\beta^{\sum_{k=1}^{i-1}t_k}R_i(t_i)+\beta^{\sum_{k=1}^{i}t_k}R_j(t_j)+...+\beta^{\sum_{k=1}^{N-1}t_k}R_N(t_N),$$

and total expected reward for permutation $\pi_{(2)}$

$$R_1(t_1)+\beta^{t_1}R_2(t_2)+...+\beta^{\sum_{k=1}^{i-1}t_k}R_j(t_j)+\beta^{(\sum_{k=1}^{i-1}t_k)+t_j}R_i(t_i)+...+\beta^{\sum_{k=1}^{N-1}t_k}R_N(t_N).$$

Thus, for the total expected reward from permutation $\pi_{(1)}$ to be no less than that of $\pi_{(2)}$ it must be true that

$$\beta^{\sum_{k=1}^{i-1}t_k}R_i(t_i)+\beta^{\sum_{k=1}^{i}t_k}R_j(t_j) \geq \beta^{\sum_{k=1}^{i-1}t_k}R_j(t_j)+\beta^{(\sum_{k=1}^{i-1}t_k)+t_j}R_i(t_i).$$

Rearranging gives

$$\beta^{\sum_{k=1}^{i-1}t_k}(1-\beta^{t_j})R_i(t_i) \geq \beta^{\sum_{k=1}^{i-1}t_k}(1-\beta^{t_i})R_j(t_j),$$

and so

$$\frac{R_i(t_i)}{1-\beta^{t_i}} \geq \frac{R_j(t_j)}{1-\beta^{t_j}}.$$

Thus, if we find a pair of adjacent jobs which are not ordered such that

$$\frac{R_i(t_i)}{1-\beta^{t_i}} \geq \frac{R_j(t_j)}{1-\beta^{t_j}},$$

we can improve the schedule by interchanging these two jobs. □

A single application of this result could certainly yield a schedule to improve upon the Whittle index ordering. However, we can further improve the resulting schedule by now finding the optimal time to allocate to each job for the improved ordering.

We thus take the schedule obtained from the Whittle heuristic as described above and repeatedly apply an algorithm which, at each stage, performs one of the following manoeuvres:

1. Reorder (and renumber accordingly) the jobs such that

$$\frac{R_1(t_1)}{1-\beta^{t_1}} \geq \frac{R_2(t_2)}{1-\beta^{t_2}} \geq \ldots \geq \frac{R_N(t_N)}{1-\beta^{t_N}},$$

   (with allocated processing times remaining unchanged).

2. For the given ordering, to each job j allocate an alternative member of $Min(j)$, with this time chosen according to step 2 of the Whittle heuristic (equation (2.3.8)).

We will call this second heuristic the **'improved Whittle index heuristic'**.

At each step, the heuristic alternates between either reordering the jobs or allocating times to an ordering, and the total expected reward increases until convergence of the successive schedules produced, where a single schedule emerges. This convergence is guaranteed by the fact there are only a finite number of jobs to be processed and only a finite number of times available for each one, and so only a finite number of schedules.

2.3.1 Numerical Results

To evaluate the performance of both our proposed heuristics we present results from 400 randomly generated problems. Each problem consisted of a batch of 10 jobs where $T_j = 10$ for all jobs j in each problem. The problems were comprised of four different types (labelled A, B, C and D) with 100 problems generated for each type.

In problems of type A, all jobs yielded a unit reward when successfully completed, with the probabilities $p_j(s), 1 \leq s \leq 10$ obtained by sampling from a uniform $U(2,5)$ distribution and normalising so that they sum to one. In problems of type B all jobs also yielded a unit reward but the probabilities $p_j(s), 1 \leq s \leq 10$ were obtained by sampling from a uniform $U(0,5)$ distribution and normalising. For problems of types C and D, the probabilities were sampled as for types A and B respectively, but the rewards were obtained by sampling from a $U(0.5, 1.5)$ distribution. The discount factor $\beta = e^{-0.05}$ was used for all problems.

For each of the 400 problems, the maximum expected reward V^{opt} from the class of fixed-time schedules was calculated using dynamic programming (equation 2.2.1) as well as the total expected rewards V^{H_1} and V^{H_2} from our two proposed heuristics. Heuristic H_1 is the Whittle index heuristic and H_2 is our improved Whittle index heuristic. The percentage suboptimality $100[(V^{opt} - V^{H_i})/V^{opt}]$ of H_i was calculated for $i = 1, 2$ and then averaged over all 100 problems within each type.

Table 1.1 clearly shows that both heuristics perform strongly. They both appear to perform well when there is greater variability in the rewards associated with each job (examples C and D). We can also see that the improved Whittle heuristic is particularly close to optimal for all problem types.

Problem type	Whittle Index Heuristic (H_1)	Improved Whittle Index (H_2)
A	0.1645	0.04404
B	0.1818	0.06661
C	0.1477	0.01233
D	0.0657	0.00105

Table 1.1 Average percentage suboptimalities for the proposed heuristics.

2.4 Batches of Identical Jobs

To further understand the characteristics of optimal fixed-time schedules as described in Section 2.2 we now return to our original formulation of the problem and investigate the special case in which all jobs are identical. Thus a single server is presented with a collection of jobs which all have the same r_j and $p_j(\cdot)$, allowing us to drop the job identifier j from the notation. Clearly, in a batch of identical jobs, the order in which jobs are processed is unimportant so our task is merely to find the vector $\mathbf{t}$ of processing times. Now the DP equation becomes

$$V(n) = \max_{1 \le t \le T} \{R(t) + \beta^t V(n-1)\}, 1 \le n \le N, \tag{2.4.1}$$

where $V(n)$ is the maximal expected reward earnable from a batch of n jobs. We write $t(n)$ as the largest value of t maximising expected reward when there are n jobs waiting to be processed and $G(0)$ as the initial index, with t^* the time achieving $G(0)$.

Building on results from previous sections, we now describe some characteristics of optimal schedules for our batch problem. The following theorem gives us an idea of the properties of both the sequences $\{V(n), n \in \mathbb{Z}^+\}$ and $\{t(n), n \in \mathbb{Z}^+\}$ which yield solutions of the DP recursion above.

Theorem 2.4.1. *(a)* $\{V(n), n \in \mathbb{Z}^+\}$ *is a strictly increasing sequence with* $V(1) = R(T)$ *and*

$$\lim_{n \to \infty} V(n) = G(0);$$

(b) $\{t(n), n \in \mathbb{Z}^+\}$ *is a decreasing sequence of minimal values with* $t(1) = T$ *and*

$$\lim_{n \to \infty} t(n) = t^*.$$

Proof. We first prove part (a). From Lemma 2.2.1, $V(n)$ is clearly strictly increasing in n. It is also obvious that $V(1) = R(T)$, if there is only one job present there is no reason for the server not to give this job the full processing allowance. Suppose that

we process each of our jobs for time t^*. The total expected reward for this schedule is then

$$\begin{aligned} R(t^*) + \beta^{t^*} R(t^*) + ... + \beta^{(n-1)t^*} R(t^*) &= \frac{(1-\beta^{nt^*})}{1-\beta^{t^*}} R(t^*), \\ &= (1-\beta^{nt^*})G(0). \end{aligned}$$

Therefore, as $V(n)$ is a maximisation,

$$V(n) \geq (1-\beta^{nt^*})G(0). \tag{2.4.2}$$

We also note that the optimal schedule is such that

$$\begin{aligned} V(n) &= R(t(n)) + \beta^{t(n)} R(t(n-1)) + ... + \beta^{\sum_{k=2}^{n} t(k)} R(t(1)), \\ &= \frac{R(t(n))}{1-\beta^{t(n)}}(1-\beta^{t(n)}) + \beta^{t(n)} \frac{R(t(n-1))}{1-\beta^{t(n-1)}}(1-\beta^{t(n-1)}) + ... + \beta^{\sum_{k=2}^{n} t(k)} \frac{R(t(1))}{1-\beta^{t(1)}}(1-\beta^{t(1)}), \\ &\leq G(0)(1-\beta^{t(n)}) + \beta^{t(n)} G(0)(1-\beta^{t(n-1)}) + ... + \beta^{\sum_{k=2}^{n} t(k)} G(0)(1-\beta^{t(1)}), \\ &= G(0)(1-\beta^{\sum_{k=1}^{n} t(k)}), \\ &\leq G(0). \end{aligned} \tag{2.4.3}$$

Thus, combining inequalities (2.4.2) and (2.4.3) gives,

$$(1-\beta^{nt^*})G(0) \leq V(n) \leq G(0), \text{ for all } n.$$

Now

$$\lim_{n\to\infty} (1-\beta^{nt^*}) = 1,$$

and so

$$\lim_{n\to\infty} V(n) = G(0).$$

We now prove part (b). To prove that $t(n)$ is decreasing in n, we suppose that $t(n)$ is not decreasing in n and obtain a contradiction. Thus there exists some m such that $t(m+1) > t(m)$. From Theorem 2.2.4 we know that $t(m)$ is minimal and so from Proposition 2.2.3 must also be maximal. Hence,

$$G(0, t(m)) \geq G(0, t(m+1)),$$

and so

$$\begin{aligned} R\{t(m)\}+\beta^{t(m)}R\{t(m+1)\} &= G\{0,t(m)\}\{1-\beta^{t(m)}\}+\beta^{t(m)}G\{0,t(m+1)\}\{1-\beta^{t(m+1)}\}, \\ &\geq G\{0,t(m+1)\}\{1-\beta^{t(m+1)}\}+\beta^{t(m+1)}G\{0,t(m)\}\{1-\beta^{t(m)}\}, \\ &= R\{t(m+1)\}+\beta^{t(m+1)}R\{t(m)\}. \end{aligned}$$

Thus, a schedule which allocates processing $\{t(m), t(m+1), t(m-1), ..., t(1)\}$ yields a reward greater than or equal to one which allocates $\{t(m+1), t(m), t(m-1), ..., t(1)\}$.

If our inequality is strict then the first schedule is better, contradicting the optimality of $\{t(m+1), t(m), t(m-1), ..., t(1)\}$. If, however, the reward from both schedules is equal then the two ordered sets of time yield the same expected reward. Thus, we can infer that $\{t(m), t(m+1), ..., t(1)\}$ is optimal. This, together with the fact that $t(m+1) > t(m)$, contradicts the choice of $t(m)$ as the largest maximiser in (2.4.1) when $n=m$, as the time $t(m+1)$ could have been chosen. Therefore $t(n)$ must be decreasing in n.

From Proposition 2.2.3 and from Theorem 2.2.4, we have that

$$t(n) \in Max \Rightarrow t(n) \geq t^*, n \in \mathbb{Z}^+. \tag{2.4.4}$$

Further, from Theorem 2.2.4, all members of the sequence of optimal times are members of Min. It is also clear that $t(1)=T$. As we now know that $t(n)$ is decreasing in n, and from (2.4.4) is bounded below by t^*, then $t(n)$ must converge to some limit. We now determine this limit.

We do this by supposing that

$$\lim_{n\to\infty} t(n) = \bar{t} > t^*, \tag{2.4.5}$$

and obtaining a contradiction. From (2.4.5) it follows that $t(n) = \bar{t}$ when $n \geq \bar{N}$, for some $\bar{N} \in \mathbb{Z}^+$ and hence that

$$\begin{aligned} V(\bar{N}+m) &= R(\bar{t}))\sum_{j=0}^{m-1}\beta^{j\bar{t}} + \beta^{m\bar{t}}V(\bar{N}), \\ &= G(0,\bar{t})(1-\beta^{m\bar{t}}) + \beta^{m\bar{t}}V(\bar{N}), m \in \mathbb{N}. \end{aligned}$$

We can now infer that

$$\lim_{n\to\infty} V(n) = \lim_{m\to\infty} V(\bar{N}+m) = G(0,\bar{t}) < G(0,t^*) = G.$$

This contradicts Theorem 2.4.1(a), and thus

$$\lim_{n\to\infty} t(n) = t^*.$$

This concludes the proof. □

From Theorem 2.4.1 we now have an estimate of the maximum expected reward earnable ($G(0)$). We also know that our processing times will be decreasing and only taken from the set of minimal times. Thus, the structure of optimal solutions is simplified, as we know it will generate a sequence of decreasing times from the set Min, remembering that Min is relatively easy to calculate. The determination of optimal processing times now only requires the determination of the smallest and largest number of jobs n for which each member of the minimal set is optimal. The following proposition provides us with a bound on the range of n for which each minimal time can be optimal, using characteristics of these minimal times.

We firstly list the minimal times in ascending order as

$$t^* = t_{(1)} < t_{(2)} < ... < t_{(M)} = T \text{ where } M = |Min|.$$

We define our optimality ranges as

$$\Gamma_q = \{n; n \in \mathbb{Z}^+ \text{ and } t(n) = t_{(q)}\}, 1 \leq q \leq M,$$

and use Θ_q for the smallest integer greater than

$$\frac{\ln([G\{0,t_{(q)}\} - G\{t_{(q)}\}]/G\{0,t_{(q)}\})}{t_{(q)}\ln\beta} + 1, 1 \leq q \leq M. \tag{2.4.6}$$

Proposition 2.4.2. *(a)* $\Gamma_1 \supseteq [\Theta_1, \infty)$;

(b) $\Gamma_q \subseteq [1, (\min_{1\leq s\leq q-1} \Theta_s) - 1], 2 \leq q \leq M.$

Proof. Fix $1 \leq q \leq M-1$. By rearrangement in the above definition, (2.4.6), Θ_q is the smallest value of n such that

$$G[0, t_{(q)}](1 - \beta^{(n-1)t_{(q)}}) > G[t_{(q)}]. \tag{2.4.7}$$

From the proof of Proposition 2.2.3, we know that $G(0, t_{(q)}) > G(t_{(q)})$ and thus there must exist an n for which (2.4.7) is true. For any such n,

$$G[0, t_{(q)}](1-\beta^{(n-1)t_{(q)}}) > \frac{\beta^{-t_{(q)}}[R(0,t)-R(0,t_{(q)})]}{(1-\beta^{t-t_{(q)}})}, \text{ where } T \geq t \geq t_{(q)}+1,$$

and hence,

$$G[0, t_{(q)}](1-\beta^{(n-1)t_{(q)}})(\beta^{t_{(q)}}-\beta^t) > [R(0,t)-R(0,t_{(q)})]. \tag{2.4.8}$$

However, note that $G[0, t_{(q)}](1 - \beta^{(n-1)t_{(q)}})$ is the expected reward earned by processing each of $(n-1)$ jobs for $t_{(q)}$ time units. Therefore

$$V(n-1) \geq G[0, t_{(q)}](1-\beta^{(n-1)t_{(q)}}),$$

and so from (2.4.8)

$$V(n-1)(\beta^{t_{(q)}}-\beta^t) > [R(0,t)-R(0,t_{(q)})],$$

which rearranges to

$$R(0, t_{(q)})+\beta^{t_{(q)}}V(n-1) > R(0,t)+\beta^t V(n-1). \tag{2.4.9}$$

Thus, when $n \geq \Theta_q$, the time $t_{(q)}$ yields a higher total expected reward than that obtained using any other choice of $T \geq t \geq t_{(q)}+1$.

By taking $q=1$ in (2.4.9), from Theorem 2.4.1(b) we know that $t(n)=t_{(1)}=t^*$ for the range $n > \Theta_1$ and part (a) now follows. From inequality (2.4.9), for $2 \geq q \geq M$, $t_{(q)}$ also yields a higher expected reward than any choice of time larger than $t_{(q)}$ when $n \geq \Theta_q$. However, when n increases to $n=\Theta_s$ for $s=q-1$, the highest expected reward is now earned by allocating a smaller time, time $t_{(q-1)}$, and part (b) now follows. □

Thus Θ_q assists us in developing an optimal fixed-time schedule using the ordered set of minimal times.

2.4.1 Allocation Examples

In order to demonstrate the implications of the material discussed, and to aid understanding, examples of four different job types are presented. Each job gives a unit reward and the discount rate $\beta = e^{-0.05}$ in all cases. For job types 1 and 2, the probabilities $p(s), 1 \leq s \leq 10$, were obtained by sampling independently from a uniform $U(2,5)$ distribution and normalising. For job types 3 and 4, the probabilities $p(s), 1 \leq s \leq 10$, were obtained by sampling independently from an exponential distribution with mean 0.5 and normalising. Tables 2.4.1(a), 2.4.1(b), 2.4.1(c) and 2.4.1(d) give details of the probabilities generated, calculations for the related indices, the maximal and minimal times for each problem and also the optimality ranges obtained from equation (2.4.6). Table 2.4.2 shows the values of $V(n)$, and the times allocated (in brackets), for the corresponding problems gained using dynamic programming iteration 2.4.1.

Tables 2.4.1(a)-(d) demonstrate the variability that exists in general for optimal schedules produced for this problem type. In some cases, few of the maximal times are also minimal times, as in Table 2.4.1(a) where there is a relatively large set of maximal times but only T and t^* are minimal times. In other cases, all the maximal times are also minimal times, such as in Table 2.4.1(d). Table 2.4.1(d) also demonstrates that not all minimal time are necessarily used in an optimal schedule.

Table 2.4.2 confirms Theorem 2.4.1, showing that $V(n)$ increases in n and $t(n)$ decreases in n, with the limit t^* (the smallest minimal time in Tables 2.4.1(a)-(d)) for each of the problems presented. It also shows agreement with the bounds applied to the times allocated by the optimality range Γ_q. Both the optimality bounds in Tables 2.4.1(a)-(d) and results from Table 2.4.2 show that the limiting processing time, t^*, is reached for small values of n for these problems. In types 1, 3 and 4, t^* is allocated when there are fewer than 10 jobs to be processed, and for problem 3 it is still allocated when there are only 16 jobs.

t	1	2	3	4	5	6	7	8	9	10
p(t)	0.132	0.094	0.076	0.101	0.11	0.088	0.068	0.108	0.127	0.097
G(0,t)	2.582	2.213	1.983	1.979	2.007	1.965	1.886	1.908	1.96	1.955
G(t-1)	2.582	1.866	1.872	2.047	2.136	1.887	1.948	2.286	2.843	1.892

Maximal times $\{1, 2, 5, 6, 9, 10\}$

Minimal times	1	10
optimality ranges	$[8, \infty)$	$[1, 7]$

Table 2.4.1(a):Details for jobs of type 1

t	1	2	3	4	5	6	7	8	9	10
p(t)	0.106	0.089	0.132	0.078	0.119	0.127	0.08	0.095	0.111	0.062
G(0,t)	2.075	1.911	2.123	1.984	2.045	2.107	2.042	2.022	2.034	1.969
G(t-1)	2.123	2.149	2.58	2.089	2.395	2.469	1.851	2.002	2.158	1.214

Maximal times $\{3, 6, 7, 9, 10\}$

Minimal times	3	6	9	10
optimality ranges	$[16, \infty)$	$[7, 15]$	$[4, 6]$	$[1, 3]$

Table 2.4.1(b):Details for jobs of type 2

t	1	2	3	4	5	6	7	8	9	10
p(t)	0.097	0.072	0.247	0.032	0.125	0.08	0.013	0.029	0.019	0.287
G(0,t)	1.882	1.647	2.653	2.183	2.229	2.13	1.902	1.763	1.636	1.949
G(t-1)	2.653	3.068	4.821	1.563	2.441	1.589	1.599	2.091	2.192	5.59

Maximal times $\{3, 5, 6, 10\}$

Minimal times	3	10
optimality ranges	$[5, \infty)$	$[1, 4]$

Table 2.4.1(c):Details for jobs of type 3

t	1	2	3	4	5	6	7	8	9	10
p(t)	0.011	0.127	0.189	0.101	0.137	0.05	0.21	0.058	0.01	0.107
G(0,t)	0.206	1.313	2.066	2.045	2.159	1.985	2.243	2.126	1.952	1.963
G(t-1)	2.243	3.068	3.69	2.4	2.679	2.491	4.088	1.128	1.122	2.091

Maximal times $\{7, 8, 10\}$

Minimal times	7	8	10
optimality ranges	$[3, \infty)$	$\emptyset$	$[1, 2]$

Table 2.4.1(d):Details for jobs of type 4

n	Type 1		Type 2		Type 3		Type 4	
1	0.769	(10)	0.775	(10)	0.767	(10)	0.772	(10)
2	1.235	(10)	1.245	(10)	1.232	(10)	1.241	(10)
3	1.518	(10)	1.530	(10)	1.514	(10)	1.525	(7)
4	1.690	(10)	1.703	(9)	1.685	(10)	1.697	(7)
5	1.794	(10)	1.808	(9)	1.789	(3)	1.802	(7)
6	1.857	(10)	1.871	(9)	1.852	(3)	1.865	(7)
7	1.895	(10)	1.910	(6)	1.890	(3)	1.904	(7)
8	1.919	(1)	1.933	(6)	1.913	(3)	1.927	(7)
9	1.933	(1)	1.948	(6)	1.927	(3)	1.941	(7)
10	1.941	(1)	1.956	(6)	1.936	(3)	1.950	(7)
11	1.946	(1)	1.961	(6)	1.941	(3)	1.955	(7)
12	1.950	(1)	1.965	(6)	1.944	(3)	1.958	(7)
13	1.951	(1)	1.966	(6)	1.946	(3)	1.960	(7)
14	1.953	(1)	1.968	(6)	1.947	(3)	1.961	(7)
15	1.953	(1)	1.968	(6)	1.948	(3)	1.962	(7)

Table 2.4.2: Values of $V(n)$ and $t(n)$ obtained using DP, for the problem types 1, 2, 3 and 4.

2.5 Multiple Job Classes

In the previous section we explored the characteristics of schedules when a batch of identical jobs is to be served. We now build on those ideas, considering the case in which the jobs requiring service are now not all identical, but can be grouped together into a number of classes. This grouping of jobs is such that all members of the same class have identical characteristics. The introduction of multiple classes to the model leads us to again be concerned with the *order* in which jobs are processed, as well as the *fixed-time* allocated to each.

Some results from previous sections still hold for the multiple class case so we will describe how they apply and go on to explore some characteristics of optimal schedules. We first define our model and introduce some notation to aid analysis.

We suppose that each job belongs to one of C distinct classes and use the identifier j to indicate which class a particular job belongs to, rather than to identify the job itself. Therefore, all jobs in class j have the same associated reward r_j and probability

of successful completion in exactly t time units $p_j(t)$, where $1 \leq t \leq T_j$.

By a simple extension of Definition 2.2.1 each class also has a general index $G_j(s)$, and initial index $G_j(0)$. Following the role the initial index has played in earlier analyses, we assume that classes have distinct initial indices and number the job classes such that

$$G_1(0) > G_2(0) > ... > G_C(0) > 0. \tag{2.5.1}$$

We use vector $\mathbf{n} = (n_1, n_2, ..., n_C)$ to represent a state in which there are n_j jobs of class j awaiting processing, $1 \leq j \leq C$. The total number of jobs to be processed across all classes is $N = \sum_{j=1}^{C} n_j$. In our analysis, we shall mainly be concentrating on the characteristics of optimal schedules in relation to individual job classes. So, for ease, when looking at class k we re-write our vector of jobs as $\mathbf{n} = (\mathbf{n}_{k-1}, n_k, \mathbf{n}^{k+1})$ where $\mathbf{n}_{k-1}$ and $\mathbf{n}^{k+1}$ represent the sub-vectors $(n_1, n_2, ..., n_{k-1})$ and $(n_{k+1}, ..., n_C)$ respectively. When $k = 1$, sub-vector $\mathbf{n}_{k-1}$ is empty and when $k = C$, sub-vector $\mathbf{n}^{k+1}$ is empty. We also use the standard notation $\mathbf{0}$ to denote a vector where all components are zero and $\mathbf{e}_j$ to denote a vector with 1 as the jth component and zeros elsewhere.

Meanwhile, our fixed-time schedule is represented by the pair $(v, \mathbf{t})$ with v an ordered set of job classes and $\mathbf{t}$ a vector of allocated processing times. Thus, under schedule $(v, \mathbf{t})$ a job of class v_1 is processed for t_1, then a job of class v_2 is processed for t_2 and so on. For example, suppose $C = 2$ and $\mathbf{n} = (3, 1)$. The schedule where $v = \{1, 2, 1, 1\}$ and $\mathbf{t} = (2, 5, 3, 3)$ would mean process a job of class 1 for 2 time units, then a job of class 2 for 5 time units, before giving the remaining two jobs of class 1 3 time units each.

We write $V_{v,\mathbf{t}}(\mathbf{n})$ for the total expected reward gained from the use of schedule $(v, \mathbf{t})$, and $V(\mathbf{n})$ for the largest such reward.i.e.

$$V(\mathbf{n}) = \max_{v,\mathbf{t}} V_{v,\mathbf{t}}(\mathbf{n}).$$

The fixed-time schedule achieving the maximum expected reward is written $(\hat{v}, \hat{\mathbf{t}})$, again assuming that if two times achieve the maximum, the largest time is always chosen.

We now develop the theory for optimal fixed-time schedules when there are multiple job classes to be processed. The following proposition compares the times allocated to two jobs from a particular class, depending on where these jobs are in the overall schedule. It is an extension of Theorem 2.4.1(b) for the multiple class model, hence the proof is similar and is omitted.

Proposition 2.5.1. *In any optimal fixed-time schedule* $(\hat{v}, \hat{\mathbf{t}})$

$$\hat{v}_k = \hat{v}_l, k < l \Longrightarrow \hat{t}_k \leq \hat{t}_l.$$

Thus, a job processed earlier under the optimal fixed-time schedule will have a smaller processing time allocated to it than another job of the same class which is to be processed later in the schedule.

We also omit the proof of the following proposition as it is a direct consequence of Lemma 2.2.1 and Theorem 2.2.4. It simply asserts that the total expected reward yielded by an optimal schedule is greater for a problem $(\mathbf{n_{j-1}}, n_j+1, \mathbf{n^{j+1}})$ than for a problem $(\mathbf{n_{j-1}}, n_j, \mathbf{n^{j+1}})$, for any class j (i.e., when there are more jobs of class j present, the total expected reward is greater). It also asserts that for the multiple class model, all the times allocated to the jobs will still be members of the set of minimal times for the respective job class.

Proposition 2.5.2. *(a)* $V(\mathbf{n})$ *is strictly increasing componentwise;*
(b) For any problem $\mathbf{n}$, $\hat{t}_k \in Min(\hat{v}_k), 1 \leq k \leq \sum_{j=1}^{C} n_j$.

The results within Proposition 2.5.2 help to simplify the determination of optimal schedules, as again we can limit our search for allocated times to the set of minimal times for each job class.

The following proposition demonstrates some interesting characteristics of the value functions for large sized problems which will further aid our development of optimal fixed-time schedules.

Proposition 2.5.3. *(a) If* $\sum_{j=1}^{k-1} n_j \geq 1$ *then*

$$\lim_{n_k \to \infty} V(\mathbf{n}_{k-1}, n_k, \mathbf{n}^{k+1}) > G_k(0), \forall \mathbf{n}^{k+1}, 2 \leq k \leq C;$$

(b)

$$\lim_{n_k \to \infty} V(\mathbf{0}_{k-1}, n_k, \mathbf{n}^{k+1}) = G_k(0), \forall \mathbf{n}^{k+1}, 1 \leq k \leq C.$$

Proof. We first show that the limits in the proposition exist. From Proposition 2.5.2 $V(\mathbf{n}_{k-1}, n_k, \mathbf{n}^{k+1})$ is strictly increasing in n_k for any choice of $\mathbf{n}_{k-1}$, $\mathbf{n}^{k+1}$. Now, take an arbitrary fixed-time schedule $(v, \mathbf{t})$. The total expected reward earned is given by

$$\begin{aligned} V_{v,\mathbf{t}}(\mathbf{n}_{k-1}, n_k, \mathbf{n}^{k+1}) &= \sum_{i=1}^{N} \beta^{\sum_{m=1}^{i-1} t_m} G_{v_i}(0, t_i)(1-\beta^{t_i}) \\ &\leq \sum_{i=1}^{N} \beta^{\sum_{m=1}^{i-1} t_m} G_{v_i}(0)(1-\beta^{t_i}) \\ &\leq G_1(0)(1-\beta^{\sum_{m=1}^{i-1} t_m}) \\ &< G_1(0). \end{aligned}$$

The inequalities come from the definition of the Gittins index and from our ordering of job classes in (2.5.1), according to the values of their respective initial indices. Thus $V(\mathbf{n}_{k-1}, n_k, \mathbf{n}^{k+1})$ is bounded above by $G_1(0)$ for all values of n_k and the limit in part (a) exists.

To conclude the proof of part (a), suppose $\sum_{j=1}^{k-1} n_j \geq 1$ for some $k, 2 \leq k \leq C$. Take the fixed-time schedule $(\tilde{v}, \tilde{\mathbf{t}})$ which first processes all class 1 jobs, allocating to each t_1^* time units, then processes all class 2 jobs, allocating each of these t_2^* time

units, and so on for all job classes. This schedule yields total expected reward

$$V_{\tilde{v},\tilde{\mathbf{t}}}(\mathbf{n}_{k-1}, n_k, \mathbf{n}^{k+1}) = \sum_{i=1}^{C} \beta^{\sum_{m=1}^{i-1} n_m t_m^*} G_i(0, t_i^*)(1-\beta^{n_i t_i^*})$$
$$\geq \sum_{i=1}^{k-1} \beta^{\sum_{m=1}^{i-1} n_m t_m^*} G_i(0)(1-\beta^{n_i t_i^*}) + \beta^{\sum_{i=1}^{k-1} n_i t_i^*} G_k(0)(1-\beta^{n_k t_k^*}), \tag{2.5.2}$$

where the inequality is obtained by substituting $G_k(0)$ for the initial indices of all job classes scheduled after jobs of class k (an action permitted by the ordering in (2.5.1) such that their initial indices are smaller than $G_k(0)$).

Taking $n_k \to \infty$ in (2.5.2) gives

$$\lim_{n_k \to \infty} V(\mathbf{n}_{k-1}, n_k, \mathbf{n}^{k+1}) \geq \lim_{n_k \to \infty} V_{\tilde{v},\tilde{\mathbf{t}}}(\mathbf{n}_{k-1}, n_k, \mathbf{n}^{k+1}) > G_k(0),$$

and thus proves part (a). The proof of part (b) is similar, referring to the case where class k is the first non-empty set of jobs, so is omitted. □

Proposition 2.5.3(a) states that as long as we have enough jobs of a particular class then we have a lower bound on the total expected reward earned from an optimal fixed-time schedule, given by this job class's initial index. Further, if such class is the first non-empty class for a particular problem, then the total expected reward earned will be the value of the initial index in the limit.

The schedule used in the proof of Proposition 2.5.3 also suggests an optimal ordering of jobs for large sized problems. The ideas contained within it are formalised in the following theorem which describes how the job classes should be ordered when the conditions in Proposition 2.5.3 are met.

Theorem 2.5.4. *For each class $k \geq 2$ there exists $N_k \in \mathbb{Z}^+$ such that $n_k \geq N_k$ implies that for all choices of $\mathbf{n}_{k-1}$, $\mathbf{n}^{k+1}$ every optimal fixed-time schedule for $(\mathbf{n}_{k-1}, n_k, \mathbf{n}^{k+1})$ processes all jobs from classes 1 to $k-1$ before any jobs from classes k to C.*

Proof. Suppose that $2 \leq k \leq C$ and define

$$N_k = min\{n_k; n_k \in \mathbb{Z}^+ \text{ and } V((\mathbf{e}_j)_{k-1}, n_k, \mathbf{0}^{k+1}) > G_k(0), 1 \leq j \leq k-1\}. \tag{2.5.3}$$

From Proposition 2.5.3, N_k must exist and be finite. Now consider some problem $(\mathbf{n}_{k-1}, n_k, \mathbf{n}^{k+1})$ for which $\sum_{j=1}^{k-1} n_j \geq 1$ and $n_k \geq N_k$. We will prove our theorem by supposing an optimal fixed-time schedule $(\hat{v}, \hat{\mathbf{t}})$ exists which processes a job from one of the classes k to C before any other jobs, and obtaining a contradiction.

From (2.5.3) and Proposition 2.5.2 we have that

$$V(\mathbf{n}_{k-1}, n_k, \mathbf{n}^{k+1}) > G_k(0).$$

Now, using this result and the fact that $(\hat{v}, \hat{\mathbf{t}})$ is an optimal fixed-time schedule, we have

$$\begin{aligned} V(\mathbf{n}_{k-1}, n_k, \mathbf{n}^{k+1}) &= V_{\hat{v},\hat{\mathbf{t}}}(\mathbf{n}_{k-1}, n_k, \mathbf{n}^{k+1}) \\ &= \sum_{i=1}^{N} \beta^{\sum_{m=1}^{i-1}} G_{\hat{v}_i}(0, \hat{t}_i)(1-\beta^{\hat{t}_i}) > G_k(0). \end{aligned} \tag{2.5.4}$$

Hence there must exist a $\sigma, 1 \leq \sigma \leq N$ such that $G_{\hat{v}_\sigma}(0, \hat{t}_\sigma) > G_k(0)$.

Let s be the smallest value of σ (i.e., the earliest scheduled job) for which $G_{\hat{v}_\sigma}(0, \hat{t}_\sigma) > G_k(0)$. Then this implies that

$$G_{\hat{v}_s}(0) > G_k(0),$$

and so $\hat{v}_s$ must belong to one of the classes 1 to $k-1$. By the construction of our schedule $(\hat{v}, \hat{\mathbf{t}})$ we therefore have $s \geq 2$.

We now apply a pairwise interchange to schedule $(\hat{v}, \hat{\mathbf{t}})$, so that the position of $(\hat{v}_s, \hat{t}_s)$ is swapped with that of $(\hat{v}_{s-1}, \hat{t}_{s-2})$. The rest of the schedule remains unchanged. We then switch $(\hat{v}_s, \hat{t}_s)$ with $(\hat{v}_{s-2}, \hat{t}_{s-2})$, and so on, moving the processing requirement further towards the beginning of the schedule until a schedule is produced for which $(\hat{v}_s, \hat{t}_s)$ is the initial processing, but which otherwise leaves $(\hat{v}, \hat{\mathbf{t}})$ unchanged. We now show that this new schedule yields a greater expected reward than $(\hat{v}, \hat{\mathbf{t}})$.

The total expected reward earned from the new schedule is given by

$$G_{\hat{v}_s}(0,\hat{t}_s)(1-\beta^{\hat{t}_s}) + \beta^{\hat{t}_s}\left\{\sum_{i=1}^{s-1}\beta^{\sum_{m=1}^{i-1}\hat{t}_m}G_{\hat{v}_i}(0,\hat{t}_i)(1-\beta^{\hat{t}_i})\right\}$$
$$+\sum_{i=s+1}^{N}\beta^{\sum_{m=1}^{i-1}\hat{t}_m}G_{\hat{v}_i}(0,\hat{t}_i)(1-\beta^{\hat{t}_i}). \tag{2.5.5}$$

It follows from our choice of s that

$$G_{\hat{v}_i}(0,\hat{t}_i) \leq G_k(0), 1 \leq i \leq s-1,$$

and thus

$$\sum_{i=1}^{s-1}\beta^{\sum_{m=1}^{i-1}\hat{t}_m}G_{\hat{v}_i}(0,\hat{t}_i)(1-\beta^{\hat{t}_i}) \leq \beta^{\sum_{m=1}^{i-1}\hat{t}_m}G_k(0)(1-\beta^{\hat{t}_i}), \tag{2.5.6}$$

and also that

$$G_{\hat{v}_s}(0,\hat{t}_s)(1-\beta^{\hat{t}_s}) > G_k(0)(1-\beta^{\hat{t}_s}). \tag{2.5.7}$$

We now deduce from (2.5.6) and (2.5.7), that (2.5.5) is larger than (2.5.4) which contradicts the optimality of the schedule $(\hat{v},\hat{\mathbf{t}})$.

Thus, for $n_k \geq N_k$, an optimal fixed-time schedule would process a job of class 1 to $k-1$ before any other jobs. The above argument can be repeated to show that in an optimal schedule the second job to be processed must be of class 1 to $k-1$, and so on, until all jobs of classes 1 to $k-1$ have been processed. This concludes the proof. □

The above result applies to the case where there are a large number of one class of jobs present, say class j. It shows that in this situation all jobs of classes $i \leq j$ will be scheduled before any jobs of classes j to C. The following corollary extends this result to describe the ordering of jobs when there are a large number of jobs of each class present.

Corollary 2.5.5. *For each class $k \geq 2$ there exists $N_k \in \mathbb{Z}^+$ such that $n_k \geq N_k$, $2 \leq k \leq C$, implies that all optimal fixed-times schedules for problem* $\mathbf{n}$ *process jobs in decreasing order of their class identifier (i.e. all class* 1 *jobs are processed first, then class* 2 *jobs and so on).*

Proof. Define N_k as in (2.5.3). Then if $n_k \geq N_k$ for all classes $k, 2 \leq k \leq C$, Theorem 2.5.4 applies to all the job classes. It thus follows that an optimal schedule would process all jobs of class 1, then all jobs of class 2, etc. □

Thus, for systems containing a large number of jobs in each class, an optimal schedule will always schedule the jobs in decreasing order of their identifier. This greatly simplifies the problem. Now the ordering has been dealt with and the server only needs to search for the optimal time to allocate to each job, remembering that from Proposition 2.5.2(b), such times for job class k will be members of the set $Min(k)$.

2.5.1 Allocation Examples

In order to elucidate the implications of the results discussed, an example is now presented for the case where $C = 2$. For this example, jobs of class 1 have the same characteristics as job type 1 from Section 2.4, so that $G_1(0) = 2.582$, $t^* = 1$ and the minimal times are $\{1, 10\}$. Jobs of class 2 have the same characteristics as job type 2, so that $G_2(0) = 2.123$, $t^* = 3$ and the minimal times are $\{3, 6, 9, 10\}$. For this example, $G_1(0) > G_2(0)$ and so the results in this section apply. Table 2.5.1 shows the job class chosen for processing, and the time applied, for each two class problem (n_1, n_2).

In agreement with Proposition 2.5.2, all the times allocated are from the minimal set for the respective job class, and are decreasing monotonically in the sense of Proposition 2.5.1. For example, for the problem $(4, 3)$ a job of class 2 would be processed first, for 6 time units. This would lead to the state $(4, 2)$ in which another job of class 2 would be processed, for 9 time units. Both these allocated times are

from the minimal set and the time allocated to the job processed earlier (when there are more jobs awaiting service) is given a shorter time.

The table also shows that when $n_2 \geq 9$, it is optimal to process all jobs of class 1 before processing any jobs of class 2, consistent with Theorem 2.5.4. However, the table shows that it is not always the case that jobs of class 1 are chosen in preference to jobs of class 2, whenever jobs of class 1 are present. In fact, in many of the problems jobs of class 2 are processed before jobs of class 1. Further, for any problem with $1 \leq n_2 \leq 8$ the last job to be processed will be of class 1. This is interesting as it could easily be assumed that an optimal schedule would choose to process the job class with highest reward, although this proves not to be the case.

<table>
<tr><th>n_2 \ n_1</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>$\{7-15\}$</th></tr>
<tr><td>1</td><td>(2,10)</td><td>(2,9)</td><td>(2,9)</td><td>(2,9)</td><td>(2,9)</td><td>(2,6)</td><td rowspan="8">(1,1)
optimal in
all cases</td></tr>
<tr><td>2</td><td>(2,9)</td><td>(2,9)</td><td>(2,9)</td><td>(2,9)</td><td>(2,6)</td><td>(1,1)</td></tr>
<tr><td>3</td><td>(2,9)</td><td>(2,9)</td><td>(2,9)</td><td>(2,6)</td><td>(2,6)</td><td>(1,1)</td></tr>
<tr><td>4</td><td>(2,9)</td><td>(2,9)</td><td>(2,6)</td><td>(2,6)</td><td>(1,1)</td><td>(1,1)</td></tr>
<tr><td>5</td><td>(2,9)</td><td>(2,6)</td><td>(2,6)</td><td>(1,1)</td><td>(1,1)</td><td>(1,1)</td></tr>
<tr><td>6</td><td>(2,6)</td><td>(2,6)</td><td>(1,1)</td><td>(1,1)</td><td>(1,1)</td><td>(1,1)</td></tr>
<tr><td>7</td><td>(2,6)</td><td>(1,1)</td><td>(1,1)</td><td>(1,1)</td><td>(1,1)</td><td>(1,1)</td></tr>
<tr><td>8</td><td>(2,6)</td><td>(1,1)</td><td>(1,1)</td><td>(1,1)</td><td>(1,1)</td><td>(1,1)</td></tr>
<tr><td>$\{9-16\}$</td><td colspan="6">(1,1)
optimal in
all cases</td><td>(1,1)
optimal in
all cases</td></tr>
</table>

Table 2.5.1 The decision (class, allocated processing) taken by the optimal fixed-time schedule for state (n_1, n_2) in some two class problems.

2.6 Arriving Jobs

So far we have considered models where a server is presented with a fixed number of jobs to work through. We now apply some of the insights gained from such models to a problem in which new jobs arrive at the server while processing is being carried out in order to determine some of the characteristics of optimal schedules in this case. We suppose that all jobs to be processed are independent and have identical characteristics. We shall suppose that jobs arrive at a single server according to a Poisson arrival process with known rate λ and they each have the same associated reward r, with discount factor $\beta = e^{-\alpha}$ applied to future rewards. The probability of successful completion of any job in exactly t time units is $p(t)$, where $1 \leq t \leq T$.

The server again decides how much processing time to allocate to each job, where decision epochs are the times of job processing completion (the end of allocated time) and the times of entry of any job to an empty system, thus the server is not able to immediately respond to the arrival of new jobs to the system.

The DP recursions for the arrival case, as formulated in earlier sections, are

$$V(n) = \max_{1 \leq t \leq T} \left\{ r \sum_{s=0}^{t} e^{-\alpha s} p(s) + e^{-\alpha t} \sum_{m=0}^{\infty} \frac{(\lambda t)^m}{m!} e^{-\lambda t} V(n+m-1) \right\}, n \geq 1$$

$$V(0) = \frac{\lambda}{\alpha+\lambda} V(1).$$

However, with the introduction of arrivals to the model, when calculating $V(n)$, the computation of these recursions requires the values of $V(n+m)$ for some $m \in \mathbb{Z}^+$. Thus we introduce the following iterative scheme:

$$u_1(n) = 0, n \geq 0 \tag{2.6.1}$$

$$u_k(n) = \max_{1 \leq t \leq T} \left\{ r \sum_{s=0}^{t} e^{-\alpha s} p(s) + e^{-\alpha s} \sum_{m=0}^{\infty} \frac{(\lambda t)^m}{m!} e^{-\lambda t} u_{k-1}(n+m-1) \right\}, n \geq 1, k \geq 2 \tag{2.6.2}$$

$$u_k(0) = \frac{\lambda}{\alpha+\lambda} u_{k-1}(1), k \geq 2. \tag{2.6.3}$$

From Puterman (1994) the above value iteration equations are guaranteed to converge to $V(n)$, as $k \to \infty$. The use of equations (2.6.1)-(2.6.3) is as follows.

Starting with an arbitrary function for $k=1$ (we choose $u_1(n)=0, n \geq 0$ for ease of calculation), we calculate our value functions $u_k(n)$ for all values of n. We then calculate $u_2(n)$ for all values of n, using the relevant $u_1(n)$ previously calculated. This process is now repeated for $k=3$, where $u_2(n)$ is used, and so on for successive values of k.

At each successive iteration, we compute, for $i \in \mathbb{N}$,

$$m_k = \min_{i \in N} (V_k(i) - V_{k-1}(i))$$
$$M_k = \max_{i \in N} (V_k(i) - V_{k-1}(i)),$$

stopping when

$$0 \leq M_k - m_k \leq \varepsilon,$$

where ε (very small, around 1×10^{-6}) has been chosen to give us our desired level of accuracy.

In order to carry out the calculation, it is necessary to truncate our state space, imposing a limit on the maximum number of jobs in the system, n.

As the above method shows, the introduction of arrivals to the model changes the characteristics of the problem somewhat, however some characteristics of optimal policies discussed earlier in this chapter can still be seen. The following theorem shows that two of the major features of optimal schedules for our models still apply. That is, the total expected reward is increasing in n and that it is bounded above by $G(0)$.

Theorem 2.6.1. *Each $u_k(n)$ is increasing in n with $u_k(n) \leq G(0)$, for all n.*

Proof. This is clearly true for $u_1(n)$. It can also be easily shown that $u_k(0) < u_k(1)$. Now, for $n \geq 1$, suppose $u_{k-1}(n)$ is increasing in n, then $u_{k-1}(n) \leq u_{k-1}(n+1)$. Now

EX LIBRIS UNIVERSITY OF EDINBURGH

for $u_k(n)$: for some time t

$$u_k(n) = \max_{1\le t\le T}\left\{ r\sum_{s=0}^{t} e^{-\alpha s}p(s) + e^{-\alpha t}\sum_{m=0}^{\infty}\frac{(\lambda t)^m}{m!}e^{-\lambda t}u_{k-1}(n+m-1)\right\} \tag{2.6.4}$$

$$u_k(n+1) = \max_{1\le t\le T}\left\{ r\sum_{s=0}^{t} e^{-\alpha s}p(s) + e^{-\alpha t}\sum_{m=0}^{\infty}\frac{(\lambda t)^m}{m!}e^{-\lambda t}u_{k-1}((n+1)+m-1)\right\}. \tag{2.6.5}$$

From our inductive hypothesis, that $u_{k-1}(n) \le u_{k-1}(n+1)$, the $\frac{(\lambda t)^m}{m!}e^{-\lambda t}$ term for each m is greater in equation (2.6.5) than the term for the same m in equation (2.6.4). As $u_k(n)$ is a maximisation and all other terms are the same, it follows that $u_k(n) \le u_k(n+1)$ and the induction holds.

We now prove that $u_k(n)$ is bounded above by $G(0)$ using an induction on k. This is clearly true for $u_1(n)$. Now suppose that $u_k(n)$ is bounded above by $G(0)$ for all n. Then for some n:

$$\begin{aligned}
u_{k+1}(n) &= \max_{1\le t\le T}\left\{ r\sum_{s=0}^{t} e^{-\alpha s}p(s) + e^{-\alpha t}\sum_{m=0}^{\infty}\frac{(\lambda t)^m}{m!}e^{-\lambda t}u_k(n+m-1)\right\} \\
&\le \max_{1\le t\le T}\left\{ r\sum_{s=0}^{t} e^{-\alpha s}p(s) + e^{-\alpha t}\sum_{m=0}^{\infty}\frac{(\lambda t)^m}{m!}e^{-\lambda t}G(0)\right\} \\
&= \max_{1\le t\le T}\left\{ r\sum_{s=0}^{t} e^{-\alpha s}p(s) + e^{-\alpha t}G(0)\right\} \\
&= \max_{1\le t\le T}\left\{ \frac{(1-e^{-\alpha t})}{(1-e^{-\alpha t})} r\sum_{s=0}^{t} e^{-\alpha s}p(s) + e^{-\alpha t}G(0)\right\} \\
&= \max_{1\le t\le T}\left\{ (1-e^{-\alpha t})G(0) + e^{-\alpha t}G(0)\right\} \\
&= G(0).
\end{aligned}$$

So the result is proven, with the inequality being a consequence of our inductive hypothesis. □

From standard value iteration results (see for example Puterman (1994)), the properties of $u_k(n)$ described in Theorem 2.4.1 also apply to $V(n)$ and thus we can now make the following assertion about $V(n)$.

Theorem 2.6.2. *$V(n)$ is increasing in n with*

$$\lim_{n\to\infty} V(n) = G(0).$$

Proof. From Theorem 2.6.1 we know that $V(n)$ is increasing in n, with

$$\lim_{n\to\infty} V(n) \leq G(0). \tag{2.6.6}$$

We now need to show that

$$\lim_{n\to\infty} V(n) = G(0).$$

We suppose that t^* is the time yielding $G(0)$. We implement a schedule which allocates the time t^* to each of n jobs and this gives total expected reward

$$\begin{aligned} R(t^*)+e^{-\alpha t^*}R(t^*)+\ldots+e^{(n-1)t^*}R(t^*)+ & \text{ (rewards earned from arriving jobs)} \\ &\geq \frac{(1-e^{-\alpha n t^*})}{1-e^{-\alpha t^*}}R(t^*) \\ &= (1-e^{-\alpha n t^*})G(0). \end{aligned}$$

Thus

$$V(n) \geq (1-e^{-\alpha n t^*})G(0),$$

and now taking $n \to \infty$ yields

$$\lim_{n\to\infty} V(n) \geq G(0). \tag{2.6.7}$$

From (2.6.6) and (2.6.7) we therefore have that

$$\lim_{n\to\infty} V(n) = G(0).$$

This concludes the proof. □

Thus, with the introduction of arrivals to our model, we have not lost some of the characteristics of optimal schedules intrinsic to our earlier models. We can still make inferences about the structure of the rewards earned and also know the maximum reward earnable before implementing any schedule.

2.6.1 Numerical Examples

We now present some examples of optimal policies for giving service to jobs of the same four types as in previous sections. Table 2.6.1(a)-(d) shows the values of $V(n)$ and also the times allocated (shown in brackets) for various values of λ. The results for the batch model from section 2.4 are also included in the table to enable comparisons to be made.

The table shows $V(n)$ increasing in n for each problem. It also shows that the times allocated are decreasing in n, as in the batch model, and they all happen to be members of the minimal set for that type.

When the arrival rate, λ, increases, the table shows that, for the same value of n, the times allocated decrease and the value of $V(n)$ increases. This is clearly shown in Table 2.6.1(b) for the case where $n=1$. For the batch model, 10 units of processing are allocated and the total expected reward is 0.775. This is very similar to the model for arrivals when $\lambda=0.0001$, when 10 units of processing are allocated and the total expected reward is 0.776. However, when the arrival rate is 0.4, only 6 units of processing are allocated and the total expected reward is 2.094.

This would seem reasonable as, with more jobs arriving during processing, the server would be more conservative with their allocation of processing to any particular job than they would be in the equivalent batch problem. This would result in shorter times being allocated, with the difference more pronounced when λ is large. Also, when more jobs may arrive there will be a greater number of jobs awaiting service and so many more opportunities to earn rewards from the arriving jobs. Thus the total expected reward would be larger.

n	Batch	λ 0.0001	0.2	0.4	0.6	0.8	1
1	0.769 (10)	0.770 (10)	1.949 (10)	2.103 (10)	2.225 (1)	2.337 (1)	2.434 (1)
2	1.235 (10)	1.237 (10)	2.002 (10)	2.140 (1)	2.264 (1)	2.378 (1)	2.476 (1)
3	1.518 (10)	1.519 (10)	2.037 (1)	2.175 (1)	2.298 (1)	2.413 (1)	2.506 (1)
4	1.690 (10)	1.690 (10)	2.070 (1)	2.207 (1)	2.329 (1)	2.442 (1)	2.528 (1)
5	1.794 (10)	1.794 (10)	2.100 (1)	2.236 (1)	2.357 (1)	2.465 (1)	2.543 (1)
6	1.857 (10)	1.857 (10)	2.129 (1)	2.263 (1)	2.382 (1)	2.485 (1)	2.554 (1)
7	1.895 (10)	1.896 (10)	2.157 (1)	2.288 (1)	2.403 (1)	2.502 (1)	2.562 (1)
8	1.919 (1)	1.929 (1)	2.182 (1)	2.311 (1)	2.423 (1)	2.515 (1)	2.568 (1)
9	1.933 (1)	1.961 (1)	2.206 (1)	2.332 (1)	2.440 (1)	2.527 (1)	2.572 (1)
10	1.941 (1)	1.991 (1)	2.229 (1)	2.352 (1)	2.456 (1)	2.536 (1)	2.575 (1)
11	1.946 (1)	2.020 (1)	2.250 (1)	2.370 (1)	2.470 (1)	2.544 (1)	2.577 (1)
12	1.950 (1)	2.047 (1)	2.270 (1)	2.386 (1)	2.482 (1)	2.550 (1)	2.579 (1)
13	1.951 (1)	2.074 (1)	2.289 (1)	2.401 (1)	2.493 (1)	2.556 (1)	2.580 (1)
14	1.953 (1)	2.098 (1)	2.307 (1)	2.416 (1)	2.503 (1)	2.560 (1)	2.580 (1)
15	1.953 (1)	2.122 (1)	2.323 (1)	2.429 (1)	2.511 (1)	2.564 (1)	2.581 (1)

2.6.1(a): Total expected reward and allocated processing for jobs of type 1, for various values of arrival rates

n	Batch	λ 0.0001	0.2	0.4	0.6	0.8	1
1	0.775 (10)	0.776 (10)	2.000 (9)	2.094 (6)	2.115 (6)	2.118 (6)	2.119 (6)
2	1.245 (10)	1.246 (10)	2.072 (6)	2.115 (6)	2.121 (3)	2.123 (3)	2.123 (3)
3	1.530 (9)	1.532 (9)	2.097 (6)	2.119 (3)	2.123 (3)	2.123 (3)	2.123 (3)
4	1.703 (9)	1.714 (9)	2.106 (6)	2.122 (3)	2.123 (3)	2.123 (3)	2.123 (3)
5	1.808 (9)	1.830 (9)	2.110 (3)	2.123 (3)	2.123 (3)	2.123 (3)	2.123 (3)
6	1.871 (9)	1.904 (9)	2.114 (3)	2.123 (3)	2.123 (3)	2.123 (3)	2.123 (3)
7	1.910 (6)	1.957 (6)	2.116 (3)	2.123 (3)	2.123 (3)	2.123 (3)	2.123 (3)
8	1.933 (6)	1.996 (6)	2.118 (3)	2.123 (3)	2.123 (3)	2.123 (3)	2.123 (3)
9	1.948 (6)	2.025 (6)	2.119 (3)	2.123 (3)	2.123 (3)	2.123 (3)	2.123 (3)
10	1.956 (6)	2.046 (6)	2.120 (3)	2.123 (3)	2.123 (3)	2.123 (3)	2.123 (3)
11	1.961 (6)	2.062 (6)	2.121 (3)	2.123 (3)	2.123 (3)	2.123 (3)	2.123 (3)
12	1.965 (6)	2.074 (6)	2.122 (3)	2.123 (3)	2.123 (3)	2.123 (3)	2.123 (3)
13	1.967 (6)	2.082 (6)	2.122 (3)	2.123 (3)	2.123 (3)	2.123 (3)	2.123 (3)
14	1.967 (6)	2.089 (6)	2.122 (3)	2.123 (3)	2.123 (3)	2.123 (3)	2.123 (3)
15	1.968 (6)	2.094 (3)	2.123 (3)	2.123 (3)	2.123 (3)	2.123 (3)	2.123 (3)

2.6.1(b): Total expected reward and allocated processing for jobs of type 2, for various values of arrival rates

n	Batch	λ 0.0001	0.2	0.4	0.6	0.8	1
1	0.767 (10)	0.768 (10)	2.081 (10)	2.441 (3)	2.601 (3)	2.636 (3)	2.646 (3)
2	1.232 (10)	1.233 (10)	2.234 (3)	2.560 (3)	2.642 (3)	2.651 (3)	2.652 (3)
3	1.514 (10)	1.515 (10)	2.345 (3)	2.612 (3)	2.651 (3)	2.653 (3)	2.653 (3)
4	1.685 (10)	1.686 (10)	2.427 (3)	2.635 (3)	2.652 (3)	2.653 (3)	2.653 (3)
5	1.789 (3)	1.820 (3)	2.487 (3)	2.645 (3)	2.653 (3)	2.653 (3)	2.653 (3)
6	1.852 (3)	1.936 (3)	2.531 (3)	2.649 (3)	2.653 (3)	2.653 (3)	2.653 (3)
7	1.890 (3)	2.036 (3)	2.564 (3)	2.651 (3)	2.653 (3)	2.653 (3)	2.653 (3)
8	1.913 (3)	2.122 (3)	2.587 (3)	2.652 (3)	2.653 (3)	2.653 (3)	2.653 (3)
9	1.927 (3)	2.196 (3)	2.605 (3)	2.652 (3)	2.653 (3)	2.653 (3)	2.653 (3)
10	1.936 (3)	2.260 (3)	2.618 (3)	2.653 (3)	2.653 (3)	2.653 (3)	2.653 (3)
11	1.941 (3)	2.314 (3)	2.627 (3)	2.653 (3)	2.653 (3)	2.653 (3)	2.653 (3)
12	1.944 (3)	2.362 (3)	2.634 (3)	2.653 (3)	2.653 (3)	2.653 (3)	2.653 (3)
13	1.946 (3)	2.402 (3)	2.639 (3)	2.653 (3)	2.653 (3)	2.653 (3)	2.653 (3)
14	1.947 (3)	2.437 (3)	2.643 (3)	2.653 (3)	2.653 (3)	2.653 (3)	2.653 (3)
15	1.948 (3)	2.467 (3)	2.645 (3)	2.653 (3)	2.653 (3)	2.653 (3)	2.653 (3)

2.6.1(c): Total expected reward and allocated processing for jobs of type 3, for various values of arrival rates

n	Batch	λ 0.0001	0.2	0.4	0.6	0.8	1
1	0.773 (10)	0.774 (10)	2.106 (7)	2.230 (7)	2.241 (7)	2.242 (7)	2.243 (7)
2	1.241 (10)	1.242 (10)	2.209 (7)	2.242 (7)	2.243 (7)	2.243 (7)	2.243 (7)
3	1.525 (7)	1.538 (7)	2.234 (7)	2.243 (7)	2.243 (7)	2.243 (7)	2.243 (7)
4	1.698 (7)	1.746 (7)	2.241 (7)	2.243 (7)	2.243 (7)	2.243 (7)	2.243 (7)
5	1.802 (7)	1.893 (7)	2.242 (7)	2.243 (7)	2.243 (7)	2.243 (7)	2.243 (7)
6	1.866 (7)	1.996 (7)	2.242 (7)	2.243 (7)	2.243 (7)	2.243 (7)	2.243 (7)
7	1.904 (7)	2.069 (7)	2.243 (7)	2.243 (7)	2.243 (7)	2.243 (7)	2.243 (7)
8	1.928 (7)	2.120 (7)	2.243 (7)	2.243 (7)	2.243 (7)	2.243 (7)	2.243 (7)
9	1.942 (7)	2.156 (7)	2.243 (7)	2.243 (7)	2.243 (7)	2.243 (7)	2.243 (7)
10	1.950 (7)	2.182 (7)	2.243 (7)	2.243 (7)	2.243 (7)	2.243 (7)	2.243 (7)
11	1.955 (7)	2.200 (7)	2.243 (7)	2.243 (7)	2.243 (7)	2.243 (7)	2.243 (7)
12	1.959 (7)	2.212 (7)	2.243 (7)	2.243 (7)	2.243 (7)	2.243 (7)	2.243 (7)
13	1.961 (7)	2.221 (7)	2.243 (7)	2.243 (7)	2.243 (7)	2.243 (7)	2.243 (7)
14	1.962 (7)	2.228 (7)	2.243 (7)	2.243 (7)	2.243 (7)	2.243 (7)	2.243 (7)
15	1.962 (7)	2.232 (7)	2.243 (7)	2.243 (7)	2.243 (7)	2.243 (7)	2.243 (7)

2.6.1(d): Total expected reward and allocated processing for jobs of type 4, for various values of arrival rates

2.7 Summary

In this chapter, we studied systems in which a server is presented with a collection of jobs and, unable to observe their successful completion during processing, chooses to allocate a fixed time to each job. A schedule was sought with the aim being to maximise some measure of expected reward. A number of models were examined, including a general case where each job has different characteristics, the special case where all jobs have identical characteristics, and the case where jobs can be classified into one of a number of classes.

A calibrating index (the Gittins index) was developed for this problem, which is easily computable and only requires information about the job under consideration. It was shown that an optimal schedule contained only times from a set defined using the Gittins index (the set of minimal times). When all the jobs to be processed were identical, the times allocated were found to decrease as the number of jobs present increased, with the limiting time being the time achieving the maximum in the Gittins index. The index also defined the total expected reward available from a large number of jobs, and featured in an expression determining the range for which each minimal time is optimal.

The problem was modelled as a restless bandit, and two heuristics based on Whittle indices were produced, both easily computable. Both of the heuristics based on the Whittle index were evaluated numerically by comparing their performance to that of the optimal solution (gained using DP) in 400 randomly generated problems. Both heuristics were found to yield results within 0.2% of the optimal solution while the 'improved Whittle heuristic' was found to yield results very close to optimal (within 0.07% of optimality).

Results in this chapter provide insights into the characteristics of optimal schedules and two well performing heuristics so aiding the development of solutions to the scheduling problems discussed.

Chapter 3

Single Class Fixed-Time Schedules, Subject to Loss

3.1 Introduction

In the previous chapter we concentrated on models in which postponing waiting jobs was discouraged by the discounting of future rewards earned from them. In this chapter we now consider a scenario where the penalty for delaying service arises out of jobs being lost from the system before they have been successfully completed.

We examine a situation in which jobs arrive at, and await service from, a single server. Each job spends a finite length of time in the system, after which it is lost. This deadline on the time the job will be in the system is unknown to the server. In common with the models in the previous chapter, the server is also unable to immediately observe whether the service he or she is giving to a job is successful. Gaver et al. (2006) cited the delivery of emergency medical treatment to patients as an example of a context in which this system could arise.

Queueing systems in which jobs have known deadlines have been widely studied. See, for example, Glazebrook (1983), Jiang et al. (1996) or Lehoczky (1997). Research has also examined queueing systems in which jobs are impatient and can be lost from the system during service, as is the case for the model considered in this chapter. For example, Harrison and Zeevi (2004) and Glazebrook et al. (2004) studied systems

where impatience may be exhibited in the form of goods which may perish and become unusable after some unknown length of time. Also, Garnett et al. (2002) examined impatience in the context of call centres, where customers held in a queue may hang up if kept waiting too long, while Doytchinov et al. (2001) cited the due dates of orders in manufacturing as an example of an application of such models. However, research in this area has generally assumed that successful job completion can be immediately observed by the server. This leaves the type of system examined in this chapter largely unstudied.

Our model extends the work done by Gaver et al. (2006), studying a system in which jobs have unknown deadlines and successful service completion is unobservable. Gaver et al. (2006) developed static policies for allocating service and discussed the fact that the development of good dynamic policies for this system was a research priority. It is these dynamic policies which we develop in this chapter.

We consider a situation in which, under the above uncertainty, the server allocates a fixed period of time for each job, prior to processing. Service for this fixed time is then carried out in its entirety, at which point the job is discarded and processing allocated to the next waiting job, if any. Successive service times are chosen dynamically, dependent on queue length, with the aim of maximising throughput (the proportion of jobs which are successfully completed). It would seem reasonable for the server to take account of the number of jobs in the system when deciding how long to process the current job - allocating large times may increase the chances of successful service but can also lead to the loss of waiting jobs. When there is a large queue, avoiding the loss of many waiting jobs may become more important than ensuring completion of the current job. Thus we would expect shorter times to be allocated when the queue is large.

Following processing, the server is unable to observe the state of the job which has just been processed, so it may have been successfully completed early, have been

lost from the system during processing (either successfully completed or not), or may still have been available but required more service (been incomplete) at the end of the allotted time. The server is also unable to observe arrivals and losses from the waiting queue between service terminations.

The remainder of this chapter is comprised of seven sections. In Section 3.2 we introduce and define the service model, then in Section 3.3 we describe how to develop an approximately optimal policy using dynamic programming. However, dynamic programming in general is known to be very computationally intensive so we also propose a static Markov policy in Section 3.4 and, in Sections 3.5 and 3.6, two easily computed dynamic policies for allocating service. Section 3.7 presents the results from a numerical study and the performance of each of the policies developed is discussed. Finally, in Section 3.8 we summarise the main findings of the chapter.

3.2 The Model

Jobs enter the system according to a Poisson arrival process with rate λ. Once there, each has an exponentially distributed length of time, with mean $\frac{1}{\theta}$ ($\theta > 0$), after which it leaves the system (unless it has already been successfully served). These times of availability for service are independent for distinct jobs. The actual service times required by jobs are independently and identically distributed with known distribution function F. Thus if a job is allocated a service time t, the probability it is successfully served in this time is given by

$$p(t) \equiv \int_0^t dF(s)e^{-\theta s}ds.$$

A single server adopts a policy whereby, at each service termination the queue length, n, is observed and a fixed time T_n is given to the next waiting job. If the system is empty at a service termination, the server waits until the next arrival to the system and then allocates a time to this job. At the end of each allocated service, the

job which is being processed leaves the system, the queue length is again observed by the server and a further fixed amount of time is allocated to the next job.

If there are n jobs present and the server allocates time t to the first job, then the random variable $N(t|n)$, the number of jobs present at the end of service, is given by the sum of two independent random variables $X(t|n)$ and $Y(t)$, both taking non-negative integer values. The number of jobs present at the beginning of service which still remain after t (excluding the job chosen for service) is represented by $X(t|n)$ and follows a binomial distribution $Bin(n-1, e^{-\theta t})$. The number of arrivals to the system during t which still remain at the end of allocated service, represented by $Y(t)$, follows a Poisson distribution $Poisson(\frac{\lambda}{\theta}(1-e^{-\theta t}))$. It can easily be seen that $Y(t)$ increases stochastically in t and $X(t|n)$ decreases stochastically in t for each n.

We seek a stationary policy, i.e., a set of service times T_n, $n \geq 1$, in order to maximise average throughput.

3.3 Dynamic Programming Solution

We now present a method for approximating the optimal solution using dynamic programming methods. Stochastic dynamic programming is most straightforward when applied to systems with a finite state space, a finite set of available actions in each state and such that these actions are chosen at discrete times, equally spaced. However, our problem's state space covers the countably infinite set of positive integers and the action space is uncountably infinite, with allocated times taking positive real values.

This presents difficulties associated with the sheer number of calculations required at each stage of the DP iteration as a large number of jobs may be present, and also with how to find the optimal continuous time to allocate to the current job. Hence, standard dynamic programming methods require modification. We now describe a

method taken from Tijms (1994).

In the proposal from Tijms (1994), we replace our continuous-time model with one in discrete-time. We imagine that the system evolves in discrete intervals δ (small) so that each decision concerns how many blocks, d, of length δ to allocate to each job. It is also necessary to truncate both the state space and the action space, imposing a maximum on the number of jobs allowed to be present and on the number of blocks of δ which may be allocated to any job. We use N for the maximum number of jobs and D for the value such that the set of possible allocated service times is $(0, \delta, 2\delta, ...D\delta)$. To obtain a good approximation, N and D need to be sufficiently large.

When considering the choice of N, we note that while our model imposes no maximum on the number of jobs allowed to queue, we can place a stochastic upper bound on the number of jobs in the system. Consider an equivalent system in which no service is given. The steady state distribution for the number of jobs in such a system is known to be Poisson with mean $(\frac{\lambda}{\theta})$ and so the maximum queue length is unlikely to exceed $\frac{\lambda}{\theta}+3\sqrt{\frac{\lambda}{\theta}}$. Thus we use this as a lower bound for N. Such a value of N would be satisfactory for all service policies.

The approximating discrete-time model allows us to use value iteration to recursively calculate a value function for the problem. We thus obtain an optimal solution for the discrete-time model. By choosing our δ to be sufficiently small, the candidate times in our discrete-time model can approximate the set of positive real values well enough and we can obtain values from this discrete-time model which closely approximate the throughputs for our original continuous-time model.

We will now describe the value iteration method for the discrete-time model before discussing how it is used to approximate the continuous-time model. The

value iteration equations for the discrete-time model are

$$V_k(n) = \max_{d\in\{1,2,\ldots D\}} \left[\int_0^{d\delta} f(s)e^{-\theta s}ds + \sum_{x=0}^{\infty} P(x|n,d)V_{k-d}(x) \right], n\geq 1, k\geq D$$

$$V_k(0) = \sum_{x=0}^{\infty} \frac{1}{x!} \left[\frac{\lambda}{\theta}\{1-e^{-\theta\delta}\} \right]^x e^{-\frac{\lambda}{\theta}\{1-e^{-\theta\delta}\}} V_{k-1}(x), k\geq D$$

$$V_k(n) = 0, \quad \text{for all} \quad k<D, n\geq 0,$$

where

$$P(x|n,d) = \sum_{r=0}^{min[n-1,x]} \binom{n-1}{r} (e^{-\theta d\delta})^r (1-e^{-\theta d\delta})^{n-1-r} \frac{1}{(x-r)!} \left\{ \frac{\lambda}{\theta}(1-e^{-\theta d\delta}) \right\}^{x-r} exp\left[\frac{-\lambda}{\theta}(1-e^{-\theta d\delta}) \right]. \quad (3.3.1)$$

The quantity $V_k(n)$ can be seen as the maximum total expected reward earnable when the current state is n and there are k periods left in the time horizon. It is composed of the probability that the current job will be completed in the allocated time, $\int_0^{d\delta} dF(s)e^{-\theta s}ds$, plus the sum of maximum expected rewards earnable when in each state x with $k-d$ periods to the time horizon, multiplied by the conditional probability that the system will be in state x at the end of the currently allocated service, which is $P(x|n,d)$. The notation reflects that fact that the probability that the system is in state x is conditional upon the state of the system, n, when service commences and also the time $d\delta$ allocated.

The system can be in state x if, for some r, r of the $n-1$ jobs in the queue are remaining in the system at the service termination and $x-r$ new jobs arrive and remain in the system during service. Hence, the first part of (3.3.1) is the binomial probability that r of the n jobs remain in the system at time $d\delta$, given that their loss rate is θ. The second part of (3.3.1) is the probability that $(x-r)$ jobs arrive and remain in the system during $d\delta$, under the poisson process.

If there are no jobs in the system at a service termination, the value $V_k(0)$ applies. In this situation, the system is observed after the interval δ, during which new jobs

may have arrived. The value function $V_k(0)$ is therefore the sum of the probabilities that x jobs arrive and remain in the system in the time δ, multiplied by the maximum expected rewards earnable from this point.

As the value iteration method we have described above requires values of $V_{k-d}(x)$ for various values of d, in order to optimise over $d \in \{1, 2, ..., D\}$, we need to set initial values for this. Hence, for convenience we set $V_k(n) = 0$, for all n and for all $k < D$.

We now describe how the value iteration algorithm is implemented. Starting with $k = D$, we calculate $V_k(n)$ for each $n \leq N$, recording both this and the value of d which provides the maximum. We now repeat the calculations for increasing values of k. As k increases, the values of $V_k(n) - V_{k-1}(n)$ for each n will converge to some limit for each n and at this limit we can obtain the solution for our discrete-time problem. To assess this convergence, for each k we calculate the differences between the value functions in successive iterations, $V_k(i) - V_{k-1}(i)$, for all $i \in \{0, 1, ..., N\}$. We then calculate

$$m_k = \min_{i \in \{0,1,...,N\}} (V_k(i) - V_{k-1}(i))$$
$$M_k = \max_{i \in \{0,1,...,N\}} (V_k(i) - V_{k-1}(i)),$$

and consider the difference, $M_k - m_k$. We repeat these calculations for increasing values of k until we are at some K for which

$$0 \leq M_K - m_K \leq \varepsilon$$

where ε (very small, around 1×10^{-6}) is chosen to give us some desired level of accuracy. We then obtain an estimate of average throughput rate, g, for the optimal policy from the identity:

$$\lim_{k \to \infty} [V_k(i) - V_{k-1}(i)] = \frac{g}{\delta}.$$

As mentioned earlier, with δ sufficiently small our solution approximates the solution for the continuous-time model. This choice of δ to approximate the continuous-time model is carried out by setting some value of δ, performing the value iteration

then repeating with $\frac{\delta}{2}$ and comparing the estimate of g which results. This is repeated until there is negligible difference in the estimates obtained (within around 6 d.p.).

Computation takes around 2 hours for a relatively large choice of δ and each time δ is halved, the number of calculations necessary at each iteration is doubled in order to search the same range of service times. This increases computation time but is necessary to be able to approximate the optimal throughput rates from the continuous-time model. As the process may need to be repeated for a number of values of δ, this can be a lengthy and laborious process.

The above dynamic programming algorithm has obvious limitations and we seek simpler and more efficient ways of obtaining good solutions to our problem.

3.3.1 Allocation Examples

Tables 3.3.1(a) and 3.3.1(b) include some examples of the allocated service times produced by the dynamic programming algorithm for a range of problems in which the actual service times are independently taken from a gamma distribution, $\Gamma(r, \nu)$ with probability density function,

$$f(t, r, \nu) = \frac{\nu^r t^{r-1} e^{-\nu t}}{\Gamma(r)}, t \geq 0,$$

where $\Gamma(\cdot)$ is the gamma function satisfying $\Gamma(r) = (r-1)!, r \in \mathbb{Z}^+$.

In these examples, the times, T_n, are decreasing in queue length n, as we would have expected. When there are a large number of jobs awaiting service, it would seem reasonable for the server to allocate a shorter time to the current job. The successive decreases in allocated service times are large for small n (up to 4) and the values rapidly converge to a limit for larger values of n (i.e, T_n is roughly decreasing convex in n). For example, in Table 3.3.1(a) for $r = 2$, in the first column where $\lambda = 0.25, \nu = 0.3$ and $\theta = 0.1$, when 1 job is present, 6.500 time units are allocated. This decreases to 5.675 units when 2 jobs are present, a difference of nearly 1 time

unit. However, in the situation where there are 7 jobs in the system, 4.830 units are allocated and this decreases to 4.785 units if there are 8 jobs present, a difference of only 0.045 time units.

Also, in these examples the values of T_n decrease as either θ or λ increase (for fixed values of n and the other parameters). These findings also agree with our intuition. A faster arrival rate would lead to more jobs arriving at the system, so a need to give each job less time. A higher value of θ means that the average length of time each job is willing to spend in the system is smaller and there is a greater incentive to serve waiting jobs sooner.

For decreasing values of the mean service time, r/ν, the times allocated also decrease. For example, in Table 3.3.1(b), in the first column where $\lambda = 1/4$ and $\theta = 0.1$, for $\Gamma(2, 0.3)$ service times (so the mean service time is 1.667) we see that when there is 1 job in the system, 2.822 time units are allocated. Keeping other parameters unchanged but choosing $r = 1/8$ (so the mean service time is 0.417) we see in Table 3.3.1(c) that when there is 1 job in the system, 1.596 time units are allocated. For jobs with a smaller mean service time, less service is required so jobs can reasonably be allocated shorter times.

We seek a strongly performing heuristic which can be computed much more easily than the dynamic programming algorithm. We would expect such a heuristic to allocate times following the same broad trends as observed in our dynamic programming algorithm.

	r=2								
(λ, ν)	(0.25,0.3)			(0.9,0.3)			(0.9,0.8)		
n \ θ	0.1	0.2	0.3	0.1	0.2	0.3	0.1	0.2	0.3
1	6.500	5.480	4.770	4.000	3.850	3.355	2.470	2.310	2.160
2	5.675	4.775	4.160	4.000	3.715	3.200	2.205	2.065	1.935
3	5.265	4.335	3.705	4.000	3.645	3.105	2.110	1.960	1.820
4	5.075	4.125	3.470	3.990	3.615	3.055	2.065	1.910	1.765
5	4.965	4.025	3.370	3.980	3.600	3.030	2.045	1.885	1.740
6	4.885	3.965	3.320	3.980	3.590	3.020	2.030	1.865	1.720
7	4.830	3.920	3.295	3.980	3.590	3.010	2.025	1.855	1.705
8	4.785	3.880	3.270	3.980	3.585	3.005	2.020	1.845	1.695
9	4.750	3.850	3.245	3.980	3.585	3.005	2.015	1.840	1.690
10	4.725	3.825	3.225	3.980	3.585	3.000	2.010	1.835	1.680

	r=1								
(λ, ν)	(0.25,0.3)			(0.9,0.3)			(0.9,0.8)		
n \ θ	0.1	0.2	0.3	0.1	0.2	0.3	0.1	0.2	0.3
1	4.006	3.302	2.868	1.972	1.758	1.594	1.540	1.382	1.272
2	2.884	2.240	1.866	1.474	1.296	1.160	1.186	1.032	0.926
3	2.410	1.816	1.474	1.244	1.080	0.954	1.020	0.872	0.774
4	2.128	1.596	1.298	1.108	0.956	0.842	0.916	0.776	0.686
5	1.934	1.438	1.166	1.012	0.868	0.762	0.844	0.710	0.622
6	1.788	1.320	1.066	0.942	0.804	0.702	0.790	0.658	0.576
7	1.674	1.230	0.988	0.888	0.752	0.656	0.746	0.618	0.540
8	1.580	1.156	0.928	0.842	0.712	0.618	0.710	0.586	0.510
9	1.502	1.094	0.878	0.806	0.676	0.588	0.680	0.558	0.484
10	1.434	1.042	0.834	0.774	0.648	0.560	0.654	0.536	0.464

Table 3.3.1(a) Values of T_n obtained from the dynamic programming value iteration algorithm for $n \in [1, 10]$ in a range of problems with gamma distributed service times

	r=1/2								
(λ, ν)	(0.25,0.3)			(0.9,0.3)			(0.9,0.8)		
n \ θ	0.1	0.2	0.3	0.1	0.2	0.3	0.1	0.2	0.3
1	2.822	2.256	1.938	1.370	1.164	1.032	1.154	0.988	0.886
2	1.730	1.212	0.944	0.936	0.740	0.622	0.820	0.652	0.552
3	1.326	0.898	0.690	0.742	0.566	0.468	0.666	0.510	0.424
4	1.092	0.720	0.546	0.626	0.466	0.378	0.572	0.428	0.350
5	0.936	0.604	0.454	0.548	0.400	0.320	0.506	0.372	0.300
6	0.824	0.524	0.390	0.490	0.352	0.280	0.458	0.330	0.264
7	0.738	0.464	0.342	0.446	0.316	0.248	0.420	0.298	0.236
8	0.670	0.416	0.306	0.410	0.286	0.224	0.388	0.272	0.214
9	0.614	0.378	0.276	0.382	0.264	0.206	0.362	0.252	0.196
10	0.568	0.346	0.252	0.356	0.244	0.190	0.340	0.234	0.182

	r=1/4								
(λ, ν)	(0.25,0.3)			(0.9,0.3)			(0.9,0.8)		
n \ θ	0.1	0.2	0.3	0.1	0.2	0.3	0.1	0.2	0.3
1	2.100	1.638	1.392	1.046	0.852	0.741	0.904	0.746	0.656
2	1.093	0.709	0.528	0.654	0.476	0.381	0.586	0.433	0.351
3	0.775	0.481	0.353	0.491	0.341	0.266	0.450	0.317	0.249
4	0.603	0.364	0.263	0.399	0.268	0.205	0.371	0.252	0.195
5	0.497	0.293	0.210	0.338	0.222	0.168	0.318	0.211	0.161
6	0.424	0.246	0.175	0.295	0.190	0.142	0.280	0.182	0.137
7	0.370	0.212	0.150	0.263	0.167	0.124	0.251	0.161	0.120
8	0.329	0.187	0.131	0.238	0.149	0.110	0.228	0.144	0.107
9	0.296	0.167	0.117	0.217	0.135	0.099	0.209	0.131	0.096
10	0.270	0.151	0.105	0.200	0.123	0.090	0.194	0.120	0.088

Table 3.3.1(b) Values of T_n obtained from the dynamic programming value iteration algorithm for $n \in [1, 10]$ in a range of problems with gamma distributed service times

	r=1/8								
(λ, ν)	(0.25,0.3)			(0.9,0.3)			(0.9,0.8)		
n \ θ	0.1	0.2	0.3	0.1	0.2	0.3	0.1	0.2	0.3
1	1.596	1.218	1.026	0.814	0.641	0.549	0.715	0.574	0.497
2	0.687	0.416	0.301	0.453	0.307	0.236	0.413	0.285	0.222
3	0.449	0.261	0.185	0.319	0.204	0.152	0.297	0.194	0.146
4	0.334	0.189	0.132	0.248	0.154	0.113	0.235	0.148	0.109
5	0.267	0.148	0.103	0.205	0.124	0.090	0.196	0.120	0.087
6	0.223	0.122	0.084	0.175	0.104	0.074	0.168	0.101	0.073
7	0.191	0.104	0.071	0.153	0.090	0.064	0.148	0.087	0.063
8	0.168	0.090	0.062	0.136	0.079	0.056	0.132	0.077	0.055
9	0.149	0.080	0.055	0.123	0.070	0.050	0.119	0.069	0.049
10	0.135	0.072	0.049	0.112	0.064	0.045	0.109	0.063	0.044

Table 3.3.1(c) Values of T_n obtained from the dynamic programming value iteration algorithm for $n \in [1, 10]$ in a range of problems with gamma distributed service times

3.4 Markov Policy

In this chapter we develop two dynamic heuristics, both of which take a simple Markovian policy as their basis. Therefore, we first present the Markovian policy and prove a result for it.

Under the Markovian policy, each job is allocated a service time independently sampled from an exponential distribution with mean $\frac{1}{\mu}$. We will call this policy μ and seek the value of μ which optimises its average throughput. This proposal follows work done by Gaver et al. (2006) in which they took the inverse of the optimal value of μ, μ^* say, to create a static policy which allocates the same constant time $\frac{1}{\mu^*}$ to all jobs, irrespective of how many jobs are present in the system. This static policy was found to perform well in the class of all static policies for our problem.

Under our Markovian policy, μ, we can write the probability that a job is successfully completed during its allocated service as

$$\varphi(\mu) = \int_0^\infty dF(s)e^{-(\theta+\mu)s}ds.$$

The number of jobs present at time t, $\tilde{N}(t)$, evolves as a birth-death process, with birth (arrival) rate λ and death (service completion or loss) rate $\mu+(n-1)\theta$ when in state $n \in \mathbb{Z}^+$. The system will reach some equilibrium distribution. Standard results from queueing theory (see, for example Puterman (1994) or Mitrani (1998)) give us the equilibrium distribution for this process:

$$\Pi_n = \frac{\lambda^n}{\mu(\mu+\theta)...(\mu+(n-1)\theta)}\Pi_0, n \geq 0,$$

with

$$\Pi_0 = \left\{1 + \sum_{n=1}^{\infty} \frac{\lambda^n}{\mu(\mu+\theta)...(\mu+(n-1)\theta)}\right\}^{-1},$$

where Π_n is the long run proportion of time the system is in state n and Π_0 is the long run proportion of time that the system is empty.

Define $\gamma(\mu)$ to be the average proportion of successful completions, or throughput, under policy μ.

Then $\gamma(\mu)$ is given by the service rate μ multiplied by the probability ($\varphi(\mu)$) this service is successful, multiplied by the proportion of time the system is not empty, i.e.

$$\begin{aligned}\gamma(\mu) &= \mu\varphi(\mu)\{1 - \Pi_0\} \\ &= \mu\varphi(\mu)\left[1 - \left[1 + \sum_{n=1}^{\infty} \frac{\lambda^n}{\mu(\mu+\theta)...(\mu+(n-1)\theta)}\right]^{-1}\right]. \qquad (3.4.1)\end{aligned}$$

Our initial aim is to maximise $\gamma(\mu)$ with respect to μ, that is to find the exponential rate under which our throughput is optimal in the class of exponential service allocations. The following lemma proves that this maximum is achieved.

Lemma 3.4.1. *There exists some* ***finite*** *value, μ^*, for which*

$$\gamma(\mu^*) = \max_{\mu} \gamma(\mu).$$

Proof. Clearly, $\gamma(0) = 0$.

We also have

$$\begin{aligned}\gamma(\mu) &= \mu\varphi(\mu)\left[1-\left\{1+\frac{\lambda}{\mu}+\frac{\lambda^2}{\mu(\mu+\theta)}+\frac{\lambda^3}{\mu(\mu+\theta)(\mu+2\theta)}\cdots\right\}^{-1}\right], \\ &\leq \mu\varphi(\mu)\left[1-\left\{\frac{1}{1-\frac{\lambda}{\mu}}\right\}^{-1}\right], \\ &= \lambda\varphi(\mu).\end{aligned}$$

From the definition of $\varphi(\mu)$, we have $\varphi(\mu)\to 0$ as $\mu\to\infty$. Hence, $\gamma(\mu)\to 0$ as $\mu\to\infty$ and so $\gamma(\mu)$, being positive valued, must have a maximum which is achieved for some finite value of μ. □

We will deploy the optimal Markovian policy, μ^*, in the development of our first dynamic heuristic.

3.5 Dynamic Heuristic Developed from the Markov Policy

While the dynamic programming algorithm seeks to allocate an optimal fixed time to each job which is processed, our first dynamic heuristic uses the idea of DP policy improvement and the application of a single policy improvement step. It supposes that we only allocate some fixed time, T, to the first job in the queue, and then apply the Markovian policy μ^* to all subsequent jobs. We wish to choose this T optimally for each queue length, n and write this policy as (T, μ^*).

The proposal to focus exclusively on the optimal time allocated to the first job, when some given policy is applied to all subsequent jobs, greatly simplifies the analysis. The choice of the Markovian policy μ^* as the policy to apply to all future jobs also allows us to take advantage of results from queueing theory.

Such results enable us to evaluate the bias function associated with the Markovian policy, written $b_{\mu^*}(\cdot)$. This represents the difference in the expected

number of jobs successfully completed over an infinite horizon under Markov policy μ^* when processing begins in state n rather than in state 0 at time 0. The evaluation of $b_{\mu^*}(n)$ will be necessary in the development of our dynamic heuristic.

We will now present a formal definition of the bias function and describe the development of the dynamic heuristic. We first define the following quantities which will be required in the analyses:

Definitions

$K_{\mu^*}(n,t)$ is the expected number of successful service completions under the Markov policy μ^* in the period $[0,t)$ when there are n jobs in the system at time 0.

$K_{(T,\mu^*)}(n,t)$ is the expected number of successful service completions under the policy (T,μ^*) in the period $[0,t)$ when there are n jobs in the system at time 0.

$\overline{K}_{\mu^*}(n)$ is the expected number of successful service completions achieved from the situation where there are n jobs in the system at time 0 until the first transition into the state 0 under the Markov policy μ^*.

$\overline{T}_{\mu^*}(n)$ is the expected time taken to go from the situation where there are n jobs in the system at time 0 until the first transition into the state 0 under the Markov policy μ^*.

Definition 3.5.1. The bias $b_{\mu^*}(n)$ is defined as

$$b_{\mu^*}(n) = \lim_{t\to\infty} \left[K_{\mu^*}(n,t) - K_{\mu^*}(0,t)\right], \tag{3.5.1}$$

where the limit in (3.5.1) is guaranteed to exist and to be finite for all $n \in \mathbb{N}$.

We will now derive an expression for the bias function which will be used in future analysis. We do this by exploiting the fact that the system under the Markov policy, μ^*, evolves as a birth-death process. Therefore, we can apply results from renewal theory which yield the following closed form for the bias function.

Lemma 3.5.1. *The quantity $b_{\mu^*}(n)$ can be evaluated as*

$$b_{\mu^*}(n) = \{\mu^* p(\mu^*) - \gamma(\mu^*)\} \left[\sum_{m=1}^{n} \sum_{s=1}^{\infty} \frac{\lambda^{s-1}}{\{\mu^* + (m-1)\theta\}...\{\mu^* + (m-1+s-1)\theta\}} \right]. \tag{3.5.2}$$

Proof. Consider the queue length process $\{\tilde{N}(t), t \in \mathbb{R}^+\}$ under the Markov policy μ^*. This process evolves as a birth-death process with birth rate λ and death rate $(\mu^*+(n-1)\theta)$ when in state $n \in \mathbb{Z}^+$.

According to standard MDP theory (see, for example, Puterman (1994) or Tijms (1994)), the birth-death process 'regenerates', or 'renews', at each entry to the empty state. This regeneration property means that the behaviour of the process is probabilistically identical from each time the process enters the empty state (i.e., each time there are no jobs present in the system). Hence, the process can be seen as the sum of a number of independent blocks of service, separated by times in which the system is empty (and the server idle).

Therefore, the evolution of the process over an infinite horizon can be split into a number of independent cycles, where each cycle describes the evolution of the process between two consecutive regeneration points (including the idle time). We now consider a single cycle and so describe the evolution of the process over an infinite horizon in terms of this single cycle.

From standard theory (see, for example Tijms (1994)), we can write our bias term as

$$\begin{aligned} b_{\mu^*}(n) =& \overline{K}_{\mu^*}(n) - \gamma(\mu^*) T_{\mu^*}(n) \\ =& \sum_{s=1}^{n} \{K_{\mu^*}(s) - K_{\mu^*}(s-1)\} - \gamma(\mu^*) \sum_{s=1}^{n} \{T_{\mu^*}(s) - T_{\mu^*}(s-1)\}. \end{aligned}$$

It now remains to evaluate $\overline{K}_{\mu^*}(n)$ and $T_{\mu^*}(n)$, which we do by computing the differences $T_{\mu^*}(m)-T_{\mu^*}(m-1)$ and $\overline{K}_{\mu^*}(m)-\overline{K}_{\mu^*}(m-1)$, with the aid of results from theory on birth-death processes.

Consider the birth-death process $\{\hat{N}(t), t\in\mathbb{R}^+\}$ which starts at time 0 in state m. This process has birth rate λ in all states and death rate $\mu^*+(n-1)\theta$ when in states $n\geq m, n\in\mathbb{N}$. When in state $(m-1)$ or below the death rate is zero. Therefore, the transition probabilities for the process $\{\hat{N}(t), t\in\mathbb{R}^+\}$ are identical to those for the process $\{\tilde{N}(t), t\in\mathbb{R}^+\}$ when in states $n\geq m$. However, $\{\hat{N}(t), t\in\mathbb{R}^+\}$ regenerates upon every entry to the state $m-1$ and does not enter any state below this.

Let $Q(m-1)$ be the steady-state probability that the process $\{\hat{N}(t), t\in\mathbb{R}^+\}$ is in state $(m-1)$. By standard results from queueing theory, $Q(m-1)$ is given by

$$Q(m-1)=\left\{1+\sum_{s=1}^{\infty}\frac{\lambda^s}{\{\mu^*+(m-1)\theta\}...\{\mu^*+(m-1)\theta+(s-1)\theta\}}\right\}^{-1}. \qquad (3.5.3)$$

Moreover, by comparison with our process $\{\tilde{N}(t), t\in\mathbb{R}^+\}$, we also know that

$$Q(m-1)=\frac{E(\text{time in state } (m-1) \text{ during one cycle})}{E(\text{length one cycle})}=\frac{\lambda^{-1}}{\lambda^{-1}+\{T_{\mu^*}(m)-T_{\mu^*}(m-1)\}}.$$

Substituting from equation (3.5.3) for $Q(m-1)$, and rearranging, gives us

$$T_{\mu^*}(m)-T_{\mu^*}(m-1)=\sum_{s=1}^{\infty}\left(\frac{\lambda^{s-1}}{\{\mu^*+(m-1)\theta\}...\{\mu^*+(m-1)\theta+(s-1)\theta\}}\right).$$

This then allows us to infer that

$$\begin{aligned}K_{\mu^*}(m)-K_{\mu^*}(m-1)&=\mu^*p(\mu^*)\{T_{\mu^*}(m)-T_{\mu^*}(m-1)\}\\&=\sum_{s=1}^{\infty}\left(\frac{\lambda^{s-1}}{\{\mu^*+(m-1)\theta\}...\{\mu^*+(m-1)\theta+(s-1)\theta\}}\right).\end{aligned}$$

We have now quantified the differences $T_{\mu^*}(m)-T_{\mu^*}(m-1)$ and $\overline{K}_{\mu^*}(m)-\overline{K}_{\mu^*}(m-1)$ and so we now return to the bias and deduce that

$$\begin{aligned}b_{\mu^*}(m)&=K_{\mu^*}(m)-\gamma(\mu^*)T_{\mu^*}(m)\\&=\sum_{s=1}^{m}\{K_{\mu^*}(s)-K_{\mu^*}(s-1)\}-\gamma(\mu^*)\sum_{s=1}^{m}\{T_{\mu^*}(s)-T_{\mu^*}(s-1)\}\\&=\{\mu^*p(\mu^*)-\gamma(\mu^*)\}\left\{\sum_{s=1}^{m}\sum_{r=1}^{\infty}\frac{\lambda^{r-1}}{\{\mu^*+(s-1)\theta\}...\{\mu^*+(s-1)\theta+(r-1)\theta\}}\right\}.\end{aligned}$$

This is our required expression for $b_{\mu^*}(n)$. □

Now that we have our expression for the bias function, we describe the development of the dynamic heuristic. Our choice of T in the dynamic heuristic emerges from the following result:

Lemma 3.5.2. *The difference between the expected number of jobs successfully completed under policy (T, μ^*) and μ^* from initial state $n \in \mathbb{Z}^+$ is given by*

$$\lim_{t\to\infty} \left\{K_{(T,\mu^*)}(n,t) - K_{\mu^*}(n,t)\right\} = p(T) + E(b_{\mu^*}[\tilde{N}(T)]) - T.\gamma(\mu^*) - b_{\mu^*}(n) \quad (3.5.4)$$

where $\tilde{N}(T)$ is the sum of two independent random variables, $X(T|n) \sim Bin\big(n-1, e^{-\theta T}\big)$ and $Y(T) \sim Poisson\big(\frac{\lambda}{\theta}\{1-e^{-\theta T}\}\big)$, and where $p(T) = \int_0^T dF(s)e^{-\theta s}ds$.

Proof. Suppose at time 0, we have n jobs present and allocate service time T to the first job, then we give all future jobs service times taken independently from the $exp(\mu^*)$ distribution. At time T, there will be $\tilde{N}(T)$ jobs in the system, as described above and so the expected number of actual service completions up to some $t \geq T$ will be

$$\begin{aligned} K_{(T,\mu^*)}(n,t) =& p(T) + E\left[K_{\mu^*}(\tilde{N}(T), t-T)\right] \\ =& p(T) + E\left[K_{\mu^*}(\tilde{N}(T), t-T) - K_{\mu^*}(0, t-T)\right] - \\ & [K_{\mu^*}(0,t) - K_{\mu^*}(0,t-T)] + K_{\mu^*}(0,t). \end{aligned}$$

Thus, by subtracting $K_{\mu^*}(n,t)$ from the above equation, we have

$$\begin{aligned} K_{(t,\mu^*)}(n,t) - K_{\mu^*}(n,t) = p(T) + E\left[K_{\mu^*}(\tilde{N}(T), t-T) - K_{\mu^*}(0, t-T)\right] - \\ [K_{\mu^*}(0,t) - K_{\mu^*}(0,t-T)] - [K_{\mu^*}(n,t) - K_{\mu^*}(0,t)]. \quad (3.5.5) \end{aligned}$$

Now, according to Blackwell's Theorem (see Ross (1983)),

$$\lim_{t\to\infty} [K_{\mu^*}(0,t) - K_{\mu^*}(0,t-T)] = T.\gamma(\mu^*). \quad (3.5.6)$$

Thus, by taking $t \to \infty$ and substituting (3.5.1) and (3.5.6) into (3.5.5) we gain our required result. □

The above result quantifies the extra throughput associated with allocating the time T rather than a time from the $exp(\mu^*)$ distribution, and provides us with a means of choosing our processing time in any state n. We will simply choose the time which maximises the r.h.s. of (3.5.4). We therefore, define

$$\tilde{T}_n = arg\max_{T\geq 0}\left\{p(T) + E(b_{\mu^*}[\tilde{N}(T)]) - T.\gamma(\mu^*)\right\}, \tag{3.5.7}$$

where $\tilde{N}(T)$ is as in Lemma 3.5.2. The following is a direct result of Lemma 3.5.2 and our construction of (3.5.7). It states that the expected number of successful completions is greater under $(\tilde{T}_n, \mu^*)$ than any other choice of $(\tilde{T}, \mu^*)$.

Lemma 3.5.3. *The policy $\tilde{T}_n$ is such that*

$$\lim_{t\to\infty}\left\{K_{(\tilde{T}_n,\mu^*)}(n,t) - K_{(\tau,\mu^*)}(n,t)\right\} \geq 0, n\in\mathbb{Z}^+$$

for any choice of $\tau\in\mathbb{R}^+$.

In order to use (3.5.7), we now substitute the closed form for $b_{\mu^*}(n)$ given in (3.5.2) and obtain the following form for our dynamic heuristic.

Theorem 3.5.4. *The policy developed by allocating a fixed time to the first job, then the Markov policy μ^* to all subsequent jobs, when in state $n \in \mathbb{N}^+$ allocates $\tilde{T}_n$ to the first job such that*

$$\tilde{T}_n = arg\max_{T\geq 0}\Bigg(p(T) - T.\gamma(\mu^*) + E\Bigg[\{\mu^*\varphi(\mu^*) - \gamma(\mu^*)\}\Bigg\{\sum_{m=1}^{\tilde{N}(T)}\sum_{s=1}^{\infty}\frac{\lambda^{s-1}}{\{\mu^* + (m-1)\theta\}...\{\mu^* + (m-1)\theta + (s-1)\theta\}}\Bigg\}\Bigg]\Bigg)$$

where $\tilde{N}(T)$ is the sum of two independent random variables, $X(T|n)\sim Bin\big(n-1, e^{-\theta T}\big)$ and $Y(T) \sim Poisson\big(\frac{\lambda}{\theta}\{1-e^{-\theta T}\}\big)$, and where $p(T) = \int_0^t dF(s)e^{-\theta s}ds$ and $\varphi(\mu) = \int_0^\infty dF(s)e^{-(\theta+\mu)s}ds$.

It is guaranteed that $\tilde{T}_n$ can be achieved for some finite value of t. Clearly, as a probability, $p(t)$ converges to a finite limit as $t \to \infty$, and by definition $b_{\mu^*}(n)$ is guaranteed to be finite.

Our dynamic heuristic is implemented so that at each decision epoch the server observes the number of jobs n in the system and allocates a job $\tilde{T}_n$ units of processing.

3.5.1 Allocation Examples

For a $\Gamma(r, \nu)$ distribution, we have the probability that a job is successfully completed under policy μ^* as,

$$\varphi(\mu^*) = \left(\frac{\nu}{\theta+\mu^*+\nu}\right)^r.$$

Tables 3.5.1(a), 3.5.1(b) and 3.5.1(c) show the times allocated by this policy, for the same examples as in Section 3.3.1. The times from our heuristic dynamic policy developed from μ^* demonstrate the same general trends as for the DP algorithm described in Section 3.3. Hence, the values of $\tilde{T}_n$ are decreasing in n and also in λ and θ (for fixed values of the other parameters). The dynamic heuristic seems to allocate larger times than those obtained from the DP approximation. For example, when $\lambda = 0.25$, $\theta = 0.1$ and for a $\Gamma(2, 0.3)$ distribution, DP allocates 4.965 time units when there are 5 jobs present, while the dynamic heuristic developed in this section allocates 6.505 time units.

	r=2								
(λ, ν)	(0.25,0.3)			(0.9,0.3)			(0.9,0.8)		
n \ θ	0.1	0.2	0.3	0.1	0.2	0.3	0.1	0.2	0.3
1	7.977	6.589	5.674	5.782	4.856	4.212	3.233	2.926	2.695
2	7.371	6.098	5.262	5.669	4.739	4.097	3.027	2.743	2.529
3	6.953	5.713	4.904	5.592	4.650	4.004	2.887	2.618	2.410
4	6.681	5.441	4.626	5.540	4.583	3.931	2.793	2.535	2.329
5	6.505	5.262	4.434	5.505	4.535	3.875	2.728	2.478	2.275
6	6.390	5.149	4.310	5.480	4.499	3.833	2.683	2.439	2.238
7	6.311	5.078	4.233	5.463	4.473	3.802	2.650	2.412	2.213
8	6.256	5.033	4.187	5.450	4.453	3.778	2.625	2.392	2.195
9	6.216	5.004	4.161	5.441	4.438	3.760	2.606	2.377	2.181
10	6.185	4.985	4.147	5.434	4.427	3.747	2.591	2.365	2.171

	r=1								
(λ, ν)	(0.25,0.3)			(0.9,0.3)			(0.9,0.8)		
n \ θ	0.1	0.2	0.3	0.1	0.2	0.3	0.1	0.2	0.3
1	4.735	3.851	3.317	2.721	2.299	0.952	1.946	1.696	1.538
2	3.782	2.974	2.506	2.333	1.941	1.698	1.642	1.400	1.251
3	3.227	2.472	2.031	2.044	1.683	1.458	1.439	1.212	1.073
4	2.915	2.227	1.823	1.836	1.511	1.306	1.301	1.094	0.967
5	2.729	2.102	1.732	1.689	1.398	1.213	1.205	1.017	0.903
6	2.610	2.031	1.687	1.583	1.322	1.153	1.135	0.965	0.860
7	2.529	1.987	1.663	1.505	1.269	1.113	1.084	0.928	0.831
8	2.470	1.958	1.648	1.446	1.231	1.084	1.045	0.901	0.810
9	2.426	1.936	1.638	1.401	1.202	1.063	1.014	0.880	0.795
10	2.392	1.920	1.631	1.365	1.180	1.047	0.989	0.864	0.782

Table 3.5.1(a) Values of $\tilde{T}_n$ allocated by the dynamic heuristic developed from the Markov policy for $n \in [1, 10]$ in a range of problems with gamma distributed service times

	r=1/2								
(λ, ν)	(0.25,0.3)			(0.9,0.3)			(0.9,0.8)		
n \ θ	0.1	0.2	0.3	0.1	0.2	0.3	0.1	0.2	0.3
1	3.132	2.491	2.132	1.682	1.377	1.203	1.330	1.120	0.997
2	2.086	1.493	1.182	1.265	0.975	0.815	1.008	0.798	0.679
3	1.599	1.109	0.866	0.994	0.740	0.605	0.814	0.623	0.520
4	1.355	0.952	0.756	0.823	0.609	0.500	0.692	0.525	0.437
5	1.217	0.872	0.705	0.713	0.533	0.442	0.612	0.465	0.390
6	1.129	0.824	0.675	0.639	0.484	0.406	0.556	0.426	0.360
7	1.070	0.793	0.657	0.587	0.451	0.382	0.516	0.399	0.339
8	1.027	0.771	0.644	0.548	0.427	0.365	0.485	0.379	0.324
9	0.994	0.755	0.634	0.519	0.410	0.353	0.461	0.364	0.313
10	0.969	0.742	0.627	0.497	0.396	0.343	0.443	0.352	0.305

	r=1/4								
(λ, ν)	(0.25,0.3)			(0.9,0.3)			(0.9,0.8)		
n \ θ	0.1	0.2	0.3	0.1	0.2	0.3	0.1	0.2	0.3
1	2.219	1.727	1.467	1.170	0.931	0.802	0.976	0.797	0.698
2	1.214	0.794	0.596	0.776	0.554	0.442	0.659	0.485	0.394
3	0.846	0.540	0.407	0.560	0.382	0.297	0.492	0.346	0.273
4	0.680	0.443	0.341	0.441	0.299	0.234	0.397	0.276	0.218
5	0.589	0.393	0.308	0.369	0.253	0.200	0.339	0.236	0.188
6	0.533	0.363	0.289	0.323	0.224	0.180	0.300	0.210	0.169
7	0.495	0.343	0.276	0.291	0.205	0.166	0.272	0.192	0.156
8	0.467	0.329	0.267	0.268	0.191	0.156	0.251	0.180	0.147
9	0.446	0.318	0.261	0.250	0.181	0.149	0.235	0.170	0.140
10	0.430	0.310	0.256	0.237	0.173	0.143	0.223	0.162	0.134

Table 3.5.1(b) Values of $\tilde{T}_n$ allocated by the dynamic heuristic developed from the Markov policy for $n \in [1, 10]$ in a range of problems with gamma distributed service times

	r=1/8								
(λ, ν)	(0.25,0.3)			(0.9,0.3)			(0.9,0.8)		
n \ θ	0.1	0.2	0.3	0.1	0.2	0.3	0.1	0.2	0.3
1	1.630	1.245	1.049	0.856	0.665	0.566	0.740	0.589	0.509
2	0.717	0.436	0.315	0.492	0.328	0.251	0.436	0.300	0.233
3	0.459	0.275	0.201	0.329	0.209	0.156	0.302	0.196	0.149
4	0.353	0.216	0.161	0.248	0.157	0.118	0.233	0.150	0.114
5	0.297	0.186	0.141	0.203	0.129	0.098	0.193	0.124	0.095
6	0.262	0.168	0.129	0.174	0.112	0.086	0.166	0.108	0.083
7	0.239	0.156	0.122	0.154	0.101	0.078	0.148	0.097	0.076
8	0.222	0.147	0.116	0.140	0.093	0.073	0.135	0.089	0.070
9	0.210	0.141	0.112	0.129	0.087	0.069	0.125	0.084	0.066
10	0.200	0.136	0.109	0.121	0.082	0.065	0.117	0.079	0.063

Table 3.5.1(c) Values of $\tilde{T}_n$ allocated by the dynamic heuristic developed from the Markov policy for $n \in [1, 10]$ in a range of problems with gamma distributed service times

3.6 Constant $\frac{1}{\mu^*}$ Improvement Policy

We now propose a second dynamic heuristic which also has its roots in the Markovian policy μ^*. For this heuristic, we apply a single DP policy improvement step to the static policy proposed by Gaver et al. (2006), who proposed that the constant time $\frac{1}{\mu^*}$ be allocated to each job. This static policy was shown to give close to maximal throughput within the static class of policies.

In applying the policy improvement step we suppose that we will choose a time to allocate to the first job to be processed, and then give all future jobs the same constant time $\frac{1}{\mu^*}$. The policy improvement step seeks to find the time to allocate to the first job which then maximises throughput over an infinite horizon.

As with the policy in Section 3.5, our policy improvement step will require the evaluation of a bias function. The bias function of interest in this section is associated with the static policy, $\frac{1}{\mu^*}$. Therefore, we consider this bias function before describing the improvement step.

We first define the following quantities:

Definitions

$C_{\frac{1}{\mu^*}}(n,t)$ is the expected number of successful service completions under the static policy $\frac{1}{\mu^*}$ in the period $[0,t)$ when there are n jobs in the system at time 0.

$\overline{C}_{\frac{1}{\mu^*}}(n)$ is the expected number of successful service completions achieved from the situation where there are n jobs in the system at time 0 until the first transition into the state 0 under the policy $\frac{1}{\mu^*}$.

$\breve{T}_{\frac{1}{\mu^*}}(n)$ is the expected time taken to go from the situation where there are n jobs in the system at time 0 until the first transition into the state 0 under policy $\frac{1}{\mu^*}$.

$\hat{g}(\frac{1}{\mu^*})$ is the average number of successful completions per unit time under the constant $\frac{1}{\mu^*}$ policy.

Definition 3.6.1. The bias function $b_{\frac{1}{\mu^*}}(n)$ is defined as

$$b_{\frac{1}{\mu^*}}(n) = \lim_{t\to\infty}\left[C_{\frac{1}{\mu^*}}(n,t) - C_{\frac{1}{\mu^*}}(0,t)\right], \tag{3.6.1}$$

where the limit in (3.6.1) is guaranteed to be finite for all $n \in \mathbb{N}$.

As in Section 3.5, we use the fact that under the static policy $\frac{1}{\mu^*}$ the system regenerates upon each entry to the empty state (see, for example, Tijms (1994)) and we gain the following result.

Lemma 3.6.1. *The bias* $b_{\frac{1}{\mu^*}}(n)$ *is given by*

$$b_{\frac{1}{\mu^*}}(n) = \overline{C}_{\frac{1}{\mu^*}}(n) - \hat{g}(\frac{1}{\mu^*})\breve{T}_{\frac{1}{\mu^*}}(n). \tag{3.6.2}$$

However, in contrast to Section 3.5, an analysis as in Lemma 3.5.1 does not lead to a closed form expression for $b_{\frac{1}{\mu^*}}(n)$. Hence we evaluate it numerically. Our description uses matrix notation. We describe procedures to evaluate $\overline{C}_{\frac{1}{\mu^*}}(n)$ and $\breve{T}_{\frac{1}{\mu^*}}(n)$ and hence $b_{\frac{1}{\mu^*}}(n)$.

We first write the state transition probabilities

$$P_{nm}(t)=P\{X(t|n)+Y(t)=m\}$$
$$=\sum_{r=0}^{min[n-1,m]}\binom{n-1}{r}(e^{-\theta t})^{r}(1-e^{-\theta t})^{n-1-r}\frac{1}{(m-r)!}\left\{\frac{\lambda}{\theta}(1-e^{-\theta t})\right\}^{m-r}exp\left[\frac{-\lambda}{\theta}(1-e^{-\theta t})\right]$$

where $t\in\mathbb{R}^{+}$, $n\in\mathbb{Z}^{+}$ and $m\in\mathbb{N}$. The transition probability, $P_{nm}(t)$, is the probability that the system will be in state m at the end of allocated service time t, given that it started in state n. It is the sum over all r of the probabilities that r of the $n-1$ jobs in the system at 0 will remain at time t and that there will be precisely $m-r$ arrivals which remain in the system at the end of time t.

Now, the expected number of jobs successfully completed before the first entry to the empty state is given by the probability that the current job will be successfully completed, plus the expected proportion of jobs completed from the state of the system at time $\frac{1}{\mu^{*}}$ multiplied by the probability of a transition into this state, for all possible states. We are only concerned with the behaviour of the system during one regeneration cycle and so the empty state is considered an absorbing state. Hence the only transition allowable in this state is to remain in it. Thus we have the following recursive equations

$$\overline{C}_{\frac{1}{\mu^{*}}}(n)=p(\frac{1}{\mu^{*}})+\sum_{m=0}^{\infty}P_{nm}\left(\frac{1}{\mu^{*}}\right)\overline{C}_{\frac{1}{\mu^{*}}}(m),n\geq 1, \tag{3.6.3}$$
$$\overline{C}_{\frac{1}{\mu^{*}}}(0)=0,$$
$$\text{where } p(\frac{1}{\mu^{*}})=\int_{0}^{\frac{1}{\mu^{*}}}dF(t)e^{-\theta t}\mathrm{d}t.$$

We now introduce the stochastic matrix $\mathbf{P}=\{P_{nm}(\frac{1}{\mu^{*}})\}$ where if $n>0$, $P_{nm}(\frac{1}{\mu^{*}})$ is given above and where $P_{00}=1$. Now, by introducing the vector $\mathbf{e}$, where

$$\mathbf{e}=\begin{cases}0, i=0\\ 1, i>0\end{cases}$$

we can write the solution to recursion (3.6.3) as

$$\begin{aligned}\overline{C}_{\frac{1}{\mu^*}}(n) &= \left[p(\frac{1}{\mu^*})\left\{\mathbf{I}+\mathbf{P}+\mathbf{P^2}+...\right\}\mathbf{e}\right]\Big|_n \\ &\equiv \left[p(\frac{1}{\mu^*})\left\{\sum_{r=0}^{\infty}\mathbf{P^r}\right\}\mathbf{e}\right]\Big|_n . \end{aligned} \tag{3.6.4}$$

We also have a recursion for the time to first entry to the empty state

$$\begin{aligned}\breve{T}_{\frac{1}{\mu^*}}(n) &= \frac{1}{\mu^*}+\sum_{m=0}^{\infty}P_{nm}(\frac{1}{\mu^*})\breve{T}_{\frac{1}{\mu^*}}(m), n \geq 1 \\ \breve{T}_{\frac{1}{\mu^*}}(0) &= 0.\end{aligned} \tag{3.6.5}$$

So now, by comparison of (3.6.3) and (3.6.5), we can infer that

$$\overline{C}_{\frac{1}{\mu^*}}(n) = \mu^* p(\frac{1}{\mu^*})\breve{T}_{\frac{1}{\mu^*}}(n), n \geq 0.$$

Thus, after calculating $\overline{C}_{\frac{1}{\mu^*}}(n)$ from (3.6.4), we can also calculate $\breve{T}_{\frac{1}{\mu^*}}(n)$. We are also interested in the time of one cycle of the system (i.e., the time between two consecutive regeneration points) in order to deduce the average throughput rate under the constant $\frac{1}{\mu^*}$ policy, using results from renewal theory. The expected number of jobs successfully served per unit time during one cycle is

$$\frac{\overline{C}_{\frac{1}{\mu^*}}(1)}{\breve{T}_{\frac{1}{\mu^*}}(1)+\lambda^{-1}} \equiv \hat{g}(\frac{1}{\mu^*}).$$

We have now evaluated $\overline{C}_{\frac{1}{\mu^*}}(n)$, and $\hat{g}(\frac{1}{\mu^*})$ and so substituting them into (3.6.2) presents us with a formulation for our bias terms as

$$b_{\frac{1}{\mu^*}}(n) = \mu^* p(\frac{1}{\mu^*})\breve{T}_{\frac{1}{\mu^*}}(n)\left[\frac{\lambda^{-1}}{\breve{T}_{\frac{1}{\mu^*}}(1)+\lambda^{-1}}\right].$$

$$b_{\frac{1}{\mu^*}}(n) = \mu^* p(\frac{1}{\mu^*})\overline{C}_{\frac{1}{\mu^*}}(n)\left[\frac{\lambda^{-1}}{\overline{C}_{\frac{1}{\mu^*}}(1)+\mu^* p(\frac{1}{\mu^*})\lambda^{-1}}\right].$$

Having evaluated the bias function and throughput under the static policy $\frac{1}{\mu^*}$, we can now implement our policy improvement step.

Lemma 3.6.2. *The difference between the expected number of successfully completed jobs under policy* $(T, \frac{1}{\mu^*})$ *and* $\frac{1}{\mu^*}$ *from initial state* $n \in \mathbb{Z}^+$ *is given by*

$$\lim_{t\to\infty}\left\{C_{(T,\frac{1}{\mu^*})}(n,t) - C_{\frac{1}{\mu^*}}(n,t)\right\} = p(T) + E(b_{\frac{1}{\mu^*}}[\tilde{N}(T)]) - T.g(\frac{1}{\mu^*}) - b_{\frac{1}{\mu^*}}(n), \tag{3.6.6}$$

where $\tilde{N}(T)$ *is the sum of two independent random variables,* $X(T|n) \sim Bin\big(n-1, e^{-\theta T}\big)$ *and* $Y(T) \sim Poisson\big(\frac{\lambda}{\theta}\{1-e^{-\theta T}\}\big)$*, and where* $p(T) = \int_0^T dF(s)e^{-\theta s}ds$.

The proof of this result follows along similar lines to that of Lemma 3.5.2, with the bias terms explicitly calculated, and so is omitted. The result provides us with the means of choosing our processing time dependent upon system state n. We simply choose the time which maximises the r.h.s. of (3.6.6). We therefore define

$$\hat{T}_n = arg\max_T\left\{p(\frac{1}{\mu^*}) - \hat{g}(\frac{1}{\mu^*})T + E(b_{\frac{1}{\mu^*}}(\tilde{N}(T)))\right\}, \tag{3.6.7}$$

where $\tilde{N}(T)$ is defined as above.

For reasons similar to those following Theorem 3.5.4, the maximum in (3.6.7) must be achieved for some finite value of T.

The following theorem shows that the policy giving time $\hat{T}_n$ to the first job (when there are n jobs in the system) and then $\frac{1}{\mu^*}$ to all subsequent jobs maximises the expected number of successful job completions above all other choices of T given to the first job. It follows directly from Lemma 3.6.2 and the construction of (3.6.7).

Theorem 3.6.3. *The policy* $\hat{T}_n$ *is such that*

$$\lim_{t\to\infty}\left\{C_{(\hat{T}_n,\frac{1}{\mu^*})}(n,t) - C_{(\tau,\frac{1}{\mu^*})}(n,t)\right\} \geq 0, n\in\mathbb{Z}^+ \tag{3.6.8}$$

for any choice of $\tau \in \mathbb{R}^+$.

The following corollary is now immediate, stating that a policy allocating the time $\hat{T}_n$ to a job at each decision epoch for which there are n jobs in the system, $n \geq 1$, will yield a higher number of expected successful service completions than the static policy $\frac{1}{\mu^*}$.

Corollary 3.6.4. *The policy $\hat{T}_n$ is such that*

$$\lim_{t\to\infty}\left\{C_{\hat{T}_n}(n,t) - C_{\frac{1}{\mu^*}}(n,t)\right\} \geq 0, n\in\mathbb{Z}^+.$$

Proof. By substitution of $\tau=\hat{T}_n$ into (3.6.8) we have that

$$\lim_{t\to\infty}\left\{K_{(\hat{T}_n,\frac{1}{\mu^*})}(n,t) - K_{\frac{1}{\mu^*}}(n,t)\right\} \geq 0$$

and thus that the policy allocating $\hat{T}_n$ at the first decision epoch, and then $\frac{1}{\mu^*}$ to all subsequent jobs achieves a higher expected number of successful job completions than a policy which allocates $\frac{1}{\mu^*}$ to all jobs.

We now use standard arguments to infer that a policy whose first r decisions are made according to the policy $\hat{T}_n$ with all remaining decisions made according to $\frac{1}{\mu^*}$ will outperform a policy which allocates $\frac{1}{\mu^*}$ at all decision epochs. Now taking $r\to\infty$ we obtain the desired result. □

The above proof guarantees that the throughputs achieved by our second heuristic, which allocates times $\hat{T}_n$ at each decision epoch, will be greater than those achieved by the static policy $\frac{1}{\mu^*}$.

3.6.1 Allocation Examples

Tables 3.6.1(a) and 3.6.1(b) show the times allocated by this policy, for the same examples as earlier. They demonstrate the same general trends as for the DP algorithm and the dynamic heuristic in Section 3.5, but for all cases the $\frac{1}{\mu^*}$ improvement policy allocates smaller times to the jobs than the dynamic heuristic from Section 3.5. Hence, these times are much closer to those obtained from the DP algorithm.

	r=2								
(λ, ν)	(0.25,0.3)			(0.9,0.3)			(0.9,0.8)		
n \ θ	0.1	0.2	0.3	0.1	0.2	0.3	0.1	0.2	0.3
1	6.787	5.518	4.785	5.074	4.037	3.404	2.666	2.415	2.228
2	5.992	4.824	4.183	5.030	3.958	3.306	2.412	2.177	2.004
3	5.524	4.378	3.732	5.008	3.911	3.243	2.278	2.050	1.877
4	5.262	4.157	3.496	4.995	3.882	3.204	2.204	1.981	1.809
5	5.105	4.054	3.401	4.987	3.863	3.178	2.160	1.940	1.769
6	5.000	3.998	3.365	4.983	3.851	3.161	2.131	1.913	1.744
7	4.924	3.961	3.350	4.979	3.842	3.149	2.111	1.894	1.727
8	4.868	3.932	3.341	4.977	3.835	3.140	2.097	1.880	1.713
9	4.824	3.908	3.331	4.975	3.829	3.133	2.086	1.870	1.703
10	4.790	3.887	3.320	4.974	3.825	3.127	2.077	1.862	1.696

	r=1								
(λ, ν)	(0.25,0.3)			(0.9,0.3)			(0.9,0.8)		
n \ θ	0.1	0.2	0.3	0.1	0.2	0.3	0.1	0.2	0.3
1	4.012	3.294	2.860	2.102	1.811	1.622	1.600	1.405	1.281
2	2.886	2.231	1.859	1.615	1.355	1.191	1.245	1.053	0.935
3	2.395	1.835	1.514	1.319	1.103	0.966	1.041	0.874	0.773
4	2.155	1.685	1.418	1.143	0.970	0.858	0.919	0.776	0.691
5	2.013	1.599	1.366	1.033	0.892	0.796	0.840	0.717	0.643
6	1.921	1.543	1.327	0.960	0.841	0.756	0.785	0.677	0.611
7	1.856	1.503	1.298	0.909	0.806	0.728	0.746	0.649	0.589
8	1.808	1.474	1.277	0.872	0.780	0.709	0.717	0.628	0.573
9	1.771	1.453	1.262	0.844	0.761	0.693	0.694	0.613	0.560
10	1.742	1.436	1.250	0.823	0.746	0.682	0.676	0.600	0.550

Table 3.6.1(a) Values of $\hat{T}(n)$ allocated by the $\frac{1}{\mu^*}$ improvement policy for $n \in [1, 10]$ in a range of problems with $\Gamma(r, \nu)$ service times

	r=1/2								
(λ, ν)	(0.25,0.3)			(0.9,0.3)			(0.9,0.8)		
n \ θ	0.1	0.2	0.3	0.1	0.2	0.3	0.1	0.2	0.3
1	2.791	2.230	1.917	1.408	1.166	1.027	1.167	0.984	0.878
2	1.693	1.181	0.919	0.967	0.739	0.614	0.828	0.645	0.543
3	1.283	0.901	0.719	0.724	0.543	0.449	0.646	0.490	0.408
4	1.092	0.788	0.644	0.591	0.448	0.375	0.541	0.411	0.344
5	0.985	0.725	0.601	0.512	0.395	0.335	0.476	0.365	0.308
6	0.916	0.686	0.575	0.461	0.362	0.311	0.432	0.335	0.286
7	0.869	0.659	0.557	0.426	0.339	0.294	0.401	0.314	0.270
8	0.835	0.640	0.543	0.401	0.323	0.282	0.377	0.299	0.259
9	0.808	0.625	0.534	0.381	0.311	0.273	0.359	0.287	0.250
10	0.788	0.614	0.526	0.367	0.302	0.266	0.345	0.278	0.243

	r=1/4								
(λ, ν)	(0.25,0.3)			(0.9,0.3)			(0.9,0.8)		
n \ θ	0.1	0.2	0.3	0.1	0.2	0.3	0.1	0.2	0.3
1	2.066	1.610	1.370	1.056	0.844	0.730	0.903	0.737	0.646
2	1.058	0.681	0.506	0.660	0.468	0.370	0.582	0.423	0.340
3	0.736	0.476	0.365	0.465	0.318	0.249	0.426	0.297	0.235
4	0.596	0.396	0.311	0.365	0.251	0.199	0.342	0.238	0.189
5	0.520	0.354	0.283	0.307	0.214	0.172	0.291	0.204	0.164
6	0.472	0.328	0.265	0.270	0.191	0.155	0.258	0.182	0.148
7	0.439	0.310	0.253	0.244	0.176	0.144	0.234	0.168	0.137
8	0.416	0.298	0.245	0.226	0.165	0.136	0.217	0.157	0.129
9	0.398	0.288	0.238	0.212	0.157	0.130	0.204	0.149	0.124
10	0.384	0.280	0.233	0.201	0.150	0.126	0.193	0.143	0.119

Table 3.6.1(b) Values of $\hat{T}(n)$ allocated by the $\frac{1}{\mu^*}$ improvement policy for $n \in [1, 10]$ in a range of problems with $\Gamma(r, \nu)$ service times

	r=1/8								
(λ, ν)	(0.25,0.3)			(0.9,0.3)			(0.9,0.8)		
θ / n	0.1	0.2	0.3	0.1	0.2	0.3	0.1	0.2	0.3
1	1.563	1.194	1.007	0.809	0.630	0.537	0.708	0.563	0.487
2	0.659	0.397	0.286	0.449	0.297	0.226	0.405	0.276	0.213
3	0.423	0.256	0.189	0.296	0.188	0.142	0.277	0.180	0.136
4	0.327	0.203	0.154	0.224	0.142	0.108	0.213	0.138	0.105
5	0.277	0.176	0.135	0.183	0.118	0.091	0.177	0.114	0.088
6	0.245	0.159	0.124	0.158	0.103	0.080	0.153	0.100	0.078
7	0.224	0.148	0.117	0.140	0.093	0.073	0.136	0.090	0.071
8	0.209	0.140	0.111	0.128	0.086	0.068	0.124	0.083	0.066
9	0.197	0.134	0.107	0.118	0.080	0.064	0.115	0.078	0.062
10	0.188	0.129	0.104	0.111	0.076	0.061	0.108	0.074	0.059

Table 3.6.1(c) Values of $\hat{T}(n)$ allocated by the $\frac{1}{\mu^*}$ improvement policy for $n \in [1, 10]$ in a range of problems with $\Gamma(r, \nu)$ service times

3.7 Numerical Results

Problems were studied for a range of cases in which the jobs' actual service times were drawn from gamma distributions, and also for various arrival and loss rates. These cases were the same as those presented in Sections 3.5 and 3.6. Tables 3.7.1(a), 3.7.1(b) and 3.7.1(c) show estimates of the proportion of arriving jobs successfully completed under the five policies discussed in the chapter:

- the Markovian policy, μ^* from Section 3.4,
- the dynamic heuristic developed from the Markov policy in Section 3.5,
- the static $\frac{1}{\mu^*}$ policy allocating time $\frac{1}{\mu^*}$ to all jobs,
- the $\frac{1}{\mu^*}$ improvement policy from Section 3.6,
- the approximate DP value iteration policy from Section 3.3.

The values in the tables for the Markovian policy were calculated using (3.4.1) for the optimal value of μ^*, and for the DP policy by the value iteration algorithm

described in Section 3.3. For the other policies, the estimates of throughput were obtained by Monte Carlo simulation. Each example was simulated 200 times (with the system allowed to burn in before recording results) and the average proportion of jobs successfully completed calculated. This approach ensured the standard errors (shown in brackets) were sufficiently small to allow comparisons of the policies to be meaningful.

Before discussing the results, we note that, as suggested earlier, the DP policy described does prove to be computationally burdensome, taking in excess of 2 hours on a standard PC for each case. Both of our dynamic heuristics are much more easily computed (taking just a few seconds).

The results in Tables 3.7.1(a), 3.7.1(b) and 3.7.1(c) show that for all the policies studied the proportion of jobs successfully completed was decreasing while each of the arrival rate λ, the loss rate θ and the mean service time $\frac{r}{\nu}$ increase. None of this is surprising.

Looking at the individual policies, while the Markovian policy μ^* performs weakly, the dynamic heuristic developed from it in Section 3.5 makes large gains on it. These gains range from 14% to 44%, and are more pronounced in the examples with lower throughput rates. The throughputs from the dynamic heuristic from Section 3.5 are also within about 5% of those from the DP policy in all cases studied. The static $\frac{1}{\mu^*}$ policy provides a similar level of performance, again within 5% of the DP policy. In fact, the static policy even outperforms the dynamic heuristic from Section 3.5 in cases where a lower proportion of jobs are completed, when $r \geq 1$. It is an interesting finding that a static policy performs so strongly within the general dynamic class.

The $\frac{1}{\mu^*}$ improvement policy outperforms the above policies and generally performs strongly, yielding throughputs which are within 1% of the DP solutions for most cases. In fact, the $\frac{1}{\mu^*}$ improvement policy actually allocates times which are very close to those from the DP policy.

		r=2				
(λ, ν)	θ	μ^*	heuristic from μ^*	static $\frac{1}{\mu^*}$	$\frac{1}{\mu^*}$ improv.	DP
	0.1	0.1554	0.2157	0.2182	0.2307	0.2271
			(0.0008)	(0.0010)	(0.0018)	
(0.25,0.3)	0.2	0.1050	0.1505	0.1508	0.1552	0.1560
			(0.0015)	(0.0016)	(0.0017)	
	0.3	0.0772	0.1118	0.1142	0.1148	0.1153
			(0.0013)	(0.0015)	(0.0015)	
	0.1	0.0617	0.0735	0.0678	0.0738	0.0753
			(0.0006)	(0.0006)	(0.0006)	
(0.9,0.3)	0.2	0.0469	0.0580	0.0564	0.0590	0.0594
			(0.0006)	(0.0006)	(0.0005)	
	0.3	0.0368	0.0466	0.0464	0.0472	0.0483
			(0.0005)	(0.0005)	(0.0005)	
	0.1	0.1775	0.2256	0.2279	0.2338	0.2337
			(0.0008)	(0.0010)	(0.0008)	
(0.9,0.8)	0.2	0.1467	0.1942	0.1964	0.2019	0.2024
			(0.0008)	(0.0009)	(0.0009)	
	0.3	0.1244	0.1687	0.1703	0.1748	0.1757
			(0.0008)	(0.0009)	(0.0009)	

		r=1				
(λ, ν)	θ	μ^*	heuristic from μ^*	static $\frac{1}{\mu^*}$	$\frac{1}{\mu^*}$ improv.	DP
	0.1	0.3292	0.4493	0.4521	0.4582	0.4584
			(0.0025)	(0.0024)	(0.0028)	
(0.25,0.3)	0.2	0.2679	0.3656	0.3610	0.3703	0.3705
			(0.0024)	(0.0024)	(0.0026)	
	0.3	0.2298	0.3123	0.3128	0.3173	0.3163
			(0.0022)	(0.0025)	(0.0025)	
	0.1	0.1904	0.2309	0.2391	0.2432	0.2432
			(0.0009)	(0.0010)	(0.0010)	
(0.9,0.3)	0.2	0.1635	0.2055	0.2096	0.2139	0.2146
			(0.0009)	(0.0009)	(0.0010)	
	0.3	0.1448	0.1873	0.1891	0.1924	0.1930
			(0.0009)	(0.0010)	(0.0009)	
	0.1	0.3580	0.4691	0.4763	0.4879	0.4876
			(0.0012)	(0.0015)	(0.0013)	
(0.9,0.8)	0.2	0.3171	0.4254	0.4267	0.4344	0.4361
			(0.0011)	(0.0014)	(0.0012)	
	0.3	0.2887	0.3884	0.3911	0.3973	0.3982
			(0.0011)	(0.0013)	(0.0012)	

Table 3.7.1(a) Estimates of the proportion of jobs successfully served under five policies, for a range of problems with $\Gamma(r, \nu)$ service times.

		r=1/2				
(λ, ν)	θ	μ^*	heuristic from μ^*	static $\frac{1}{\mu^*}$	$\frac{1}{\mu^*}$ improv.	DP
	0.1	0.5214	0.6530	0.6472	0.6555	0.6598
			(0.0032)	(0.0034)	(0.0035)	
(0.25,0.3)	0.2	0.4667	0.5823	0.5794	0.5842	0.5869
			(0.0034)	(0.0034)	(0.0034)	
	0.3	0.4315	0.5396	0.5363	0.5406	0.5403
			(0.0031)	(0.0033)	(0.0032)	
	0.1	0.3901	0.4848	0.4806	0.4924	0.4924
			(0.0014)	(0.0015)	(0.0015)	
(0.9,0.3)	0.2	0.3558	0.4429	0.4381	0.4469	0.4512
			(0.0013)	(0.0014)	(0.0014)	
	0.3	0.3331	0.4194	0.4142	0.4235	0.4228
			(0.0014)	(0.0015)	(0.0015)	
	0.1	0.5515	0.6912	0.6831	0.6960	0.6966
			(0.0016)	(0.0017)	(0.0017)	
(0.9,0.8)	0.2	0.5116	0.6441	0.6359	0.6474	0.6482
			(0.0017)	(0.0018)	(0.0017)	
	0.3	0.4846	0.6112	0.6036	0.6136	0.6139
			(0.0018)	(0.0019)	(0.0018)	

		r=1/4				
(λ, ν)	θ	μ^*	heuristic from μ^*	static $\frac{1}{\mu^*}$	$\frac{1}{\mu^*}$ improv.	DP
	0.1	0.6859	0.7850	0.7792	0.7860	0.7951
			(0.0038)	(0.0038)	(0.0038)	
(0.25,0.3)	0.2	0.6460	0.7463	0.7421	0.7476	0.7463
			(0.0038)	(0.0038)	(0.0038)	
	0.3	0.6204	0.7123	0.7079	0.7120	0.7155
			(0.0039)	(0.0039)	(0.0039)	
	0.1	0.5909	0.6859	0.6765	0.6885	0.6904
			(0.0018)	(0.0019)	(0.0019)	
(0.9,0.3)	0.2	0.5601	0.6543	0.6448	0.6557	0.6555
			(0.0021)	(0.0021)	(0.0021)	
	0.3	0.5402	0.6297	0.6216	0.6299	0.6320
			(0.0019)	(0.0019)	(0.0019)	
	0.1	0.7111	0.8200	0.8108	0.8208	0.8242
			(0.0019)	(0.0020)	(0.0019)	
(0.9,0.8)	0.2	0.6799	0.7879	0.7783	0.7878	0.7891
			(0.0020)	(0.0020)	(0.0021)	
	0.3	0.6593	0.7644	0.7564	0.7647	0.7656
			(0.0018)	(0.0019)	(0.0019)	

Table 3.7.1(b) Estimates of the proportion of jobs successfully served under five policies, for a range of problems with $\Gamma(r, \nu)$ service times.

		r=1/8				
(λ, ν)	θ	μ^*	heuristic from μ^*	static $\frac{1}{\mu^*}$	$\frac{1}{\mu^*}$ improv.	DP
	0.1	0.8055	0.8786	0.8712	0.8787	0.8775
			(0.0043)	(0.0042)	(0.0043)	
(0.25,0.3)	0.2	0.7800	0.8397	0.8356	0.8402	0.8503
			(0.0042)	(0.0043)	(0.0042)	
	0.3	0.7637	0.8284	0.8242	0.8281	0.8319
			(0.0042)	(0.0040)	(0.0042)	
	0.1	0.7468	0.8191	0.8096	0.8192	0.8183
			(0.0023)	(0.0023)	(0.0023)	
(0.9,0.3)	0.2	0.7246	0.7956	0.7870	0.7960	0.7945
			(0.0021)	(0.0021)	(0.0021)	
	0.3	0.7106	0.7745	0.7697	0.7766	0.7788
			(0.0020)	(0.0021)	(0.0021)	
	0.1	0.8234	0.8931	0.8862	0.8930	0.8967
			(0.0022)	(0.0022)	(0.0022)	
(0.9,0.8)	0.2	0.8024	0.8705	0.8641	0.8706	0.8747
			(0.0024)	(0.0024)	(0.0024)	
	0.3	0.7888	0.8613	0.8544	0.8610	0.8602
			(0.0021)	(0.0021)	(0.0021)	

Table 3.7.1(c) Estimates of the proportion of jobs successfully served under five policies, for a range of problems with $\Gamma(r, \nu)$ service times.

3.8 Summary

In this chapter, we studied a service system in which jobs arrive at a single server and are prepared to wait an unknown length of time before being lost from the system. The server is unable to immediately observe whether a job has been successfully served and so gives each job some fixed amount of service according to the number of jobs present in the system.

A dynamic programming algorithm was formulated and four alternative heuristics developed. Two of these heuristics were relatively simple static policies. One allocated a time taken from an exponential distribution to each job, while the other allocated a constant time to each job. The other two heuristics dynamically allocated processing times depending on how many jobs were currently present. Each dynamic heuristic

was developed by imagining that all future jobs would be allocated some constant service as prescribed by one of the simpler heuristics described above and constructed to yield a greater reward rate than its respective static policy.

Numerical examples showed that for both the dynamic heuristics and the DP algorithm presented, the times allocated to each job decreased as the number of jobs in the system increased. In numerical study, the dynamic heuristics yielded results within 5% of the optimal solution, with the best performing heuristic yielding results within 1% of the optimal solution. The static policy which allocated the constant time $\frac{1}{\mu^*}$ to each job also performed well, yielding results within 5% of optimal. All the heuristics proposed were also more computationally efficient than the DP algorithm, suggesting that they could be practical methods for solving this type of problem.

Chapter 4

Allocation of Servers in a Multiclass Queueing system Subject to Losses

4.1 Introduction

In the previous chapter we examined a service system in which a stream of identically distributed jobs arrive at a single server and await service for some unknown time, after which they are lost from the system. We now seek to explore a multiclass model where jobs entering the system can be classified into one of a number of types.

In the model in Chapter 3, the server is unable to immediately observe successful service completion and only observes the state of the system at each service termination. We now introduce a multiclass model where a group of servers are to be allocated between job classes. Successful service is still uncertain, however, in contrast to the model in Chapter 3, the servers are able to observe events within the system (job arrivals, losses and successful completions) as they happen. For this multiclass model, jobs are members of one of a number of job classes, where each job of the same class has the same loss rate and service completion rate, as well as earning the same reward upon successful completion. Such jobs arrive in independent Poisson streams at a central group of servers, where at each point in time, each job

class is allocated one of a number of servers from the central group. The aim is to allocate the servers to the job classes such that the average reward rate earned across all job classes is maximised.

Gaver et al. (2006) investigated the allocation of service subject to jobs being lost from the system before being served, and described situations in which these systems could arise. These include medical patients who may die before being treated, and perishable goods which may become unusable before service has been successfully completed. These applications were modelled by Gaver et al. (2006) as a single class queueing system subject to loss, but could with profit be extended to include many different job classes. Boots and Tijms (1999) and Whitt (1999) have also examined queueing systems where customers may be lost during service.

More recently, Glazebrook et al. (2004) analysed a single server multiclass queueing system subject to jobs being lost during service. The model we now present is a generalisation to a group of servers of the model introduced by Glazebrook et al. (2004). For their single server system, Glazebrook et al. (2004) developed an index based heuristic where the server was allocated to the job class with highest value index and this index took account of the number of jobs of each class present in the system. We follow the methods used in developing this index, extending it to our more general model.

As previously described, the multiclass model examined in this chapter seeks to allocate a group of servers between a number of job classes. Servers allocated to the same job class work together as a team, so the more servers there are, the faster the service completion rate. The system controller is able to immediately observe the arrival, loss or successful completion of a job, and can thus observe the state of the system continuously (as it happens).

The remainder of this chapter is comprised of six sections. In Section 4.2, we formally define the model for the multiclass system. We then develop a dynamic

programming value iteration scheme for the problem in Section 4.3, and describe how this scheme can in principle be applied to obtain an optimal solution. In Section 4.4, we develop our first heuristic, a static policy allocating the available servers between job classes irrespective of how many jobs of each class are present at any time. In Section 4.5, we then develop our second heuristic by implementing a single policy improvement step on the static policy from Section 4.4 to produce a heuristic which dynamically allocates servers according to how many jobs of each class are currently present. Section 4.6 presents the results from a numerical study and the performance of each of the policies developed is discussed. Finally, in Section 4.7 we summarise the main findings of the chapter.

4.2 The Model

We have a set of jobs which can be classified into one of M different classes. Jobs from class j arrive at a service point according to a Poisson process with rate λ_j, $1 \leq j \leq M$, independently of the arrivals from other classes. Once in the system, each job has an (independent) exponentially distributed length of time available for service, with mean $(\theta_j)^{-1}$, after which it leaves the system (unless it has already been successfully served). The job will leave the system after this time, whether it is currently in service or not.

A central group of N servers ($N \in \mathbb{N}^+$) is available, to be allocated between the job classes. If the jobs of class j are allocated u_j servers they will be served at an (exponential) rate $\mu_j(u_j), 0 \leq u_j \leq N, 1 \leq j \leq M$. Note that it is natural to assume that the function $\mu_j(u_j)$ is increasing in u and is concave, reflecting a law of diminishing returns as more servers are deployed. For each successfully completed job, a reward r_j is earned.

In the model, we define our state as the vector $\mathbf{n} = (n_1, n_2, ..., n_M), n_j \in \mathbb{N}$, where n_j is the number of jobs of class j present in the system. Hence the state space of the system is $\mathbb{N}^{\mathbb{M}}$. The system controller is able to examine the system at each new event, whether this is an arrival, loss or successful service completion, and choose a new server deployment accordingly i.e., change the number of servers allocated to each job class according to how many jobs of each class are now in the system. (Recall that in previous chapters the system was only observable at the end of a period of allocated service.) Our problem is how to allocate our servers in order to maximise the total reward rate. The action space, or set of possible server deployments is given by

$$A = \{\mathbf{u}; u_j \geq 0, 1 \leq j \leq M, and \sum_{j=1}^{M} u_j = N\}$$

and we seek an optimal stationary policy, namely a map from $\mathbb{N}^{\mathbb{M}}$ to A which maximises the reward rate.

The system as modelled forms a Markov Decision Process (MDP). Therefore, standard MDP theory applies to this case and we can assume that an optimal policy exists which is stationary (allocates the same proportion of servers to each job class each time the system is in the same state). Thus, in our search for policies we need only consider stationary policies as is indicated above.

As a model incorporating losses from the system through the loss rates θ_i, a stable system is guaranteed. Hence, under stationary policies, over the long run some natural equilibrium distribution will be reached for the number of jobs of each class present. This would be so even were no service to be given.

4.3 Dynamic Programming Solution

We first develop a dynamic programming algorithm which is guaranteed to yield the optimal solution for this model, using value iteration. A requirement of standard value iteration is that decision epochs (the instances when decisions on server allocation are made) occur at intervals, the spaces between which are equal in expectation. However, in our model, decision epochs do not arrive at equal rates. Rather, in state $\mathbf{n}$ under action $\mathbf{u}$ the rate at which the next event occurs will be

$$\sum_{j=1}^{M} \lambda_j + \sum_{j=1}^{M} \mu_j(u_j) + \sum_{j=1}^{M} \theta_j n_j,$$

which varies with both $\mathbf{n}$ and $\mathbf{u}$.

We therefore introduce extra, fictitious 'null' events between the actual events, a uniformization method epoused by Tijms (1994). At such null events, the servers observe the system, but as the state has actually remained unchanged the decision is to continue with the current service vector. This extra option to observe the system more regularly between actual arrivals, losses or service completions enables us to model the system as one in which events occur at a constant rate and hence develop a standard value iteration algorithm.

The introduction of null events is possible as job arrivals, losses and service distributions are all exponential and so exhibit the memoryless property. Thus, the time to the next event from a null event is probabilistically identical to the equivalent time from the last actual event.

We will now describe the value iteration equations including null events. Firstly, we uniformize the time between decision epochs by calculating a bound on the maximum event rate, expressed by the following equation:

$$\sum_{i=1}^{M} \lambda_i + \sum_{i=1}^{M} \mu_i(N) + \sum_{i=1}^{M} \theta_i Q_i = A.$$

In this expression, $\mu_i(N)$ is the service rate for job class i if all servers are allocated to this class, Q_i is an imposed maximum queue length for job i and $\theta_i Q_i$ is the loss

rate from class i when there are Q_i jobs of class i present. Therefore, the reciprocal of A defines the mean inter-event time post uniformization for use in the value iteration equations.

Note There is no restriction on the number of jobs of each class allowed to queue for service. However, in order to calculate A, we need to choose a value for the maximum queue length Q_i, for all $1 \leq i \leq M$. We use $Q_i \gg \frac{\lambda_i}{\theta_i} + 2\sqrt{\frac{\lambda_i}{\theta_i}}$ as, even in the absence of service, the queue for class i is unlikely to exceed this. This value of Q_i is used to truncate the state space when performing computations using value iteration.

We now introduce the value iteration equations as follows:

$$
\begin{aligned}
V_0(\mathbf{n}) =& 0, \text{for all } \mathbf{n}, \\
V_t(\mathbf{n}) =& \max_u \Bigg\{ \frac{\sum_{i=1}^M \lambda_i V_{t-1}(\mathbf{n}+\mathbf{1}^i)}{A} + \frac{\sum_{i=1}^M \mu_i(u_i) I_i \{r_i + V_{t-1}(\mathbf{n}-\mathbf{1}^i)\}}{A} \\
&+ \frac{\sum_{i=1}^M \theta_i n_i V_{t-1}(\mathbf{n}-\mathbf{1}^i)}{A} + \frac{\sum_{i=1}^M \{\mu_i(N) - \mu_i(u_i) I_i\} V_{t-1}(\mathbf{n})}{A} \\
&+ \frac{\sum_{i=1}^M \theta_i (Q_i - n_i) V_{t-1}(\mathbf{n})}{A} \Bigg\}, \qquad (4.3.1) \\
=& \frac{\sum_{i=1}^M \{\lambda_i V_{t-1}(\mathbf{n}+\mathbf{1}^i) + \theta_i n_i V_{t-1}(\mathbf{n}-\mathbf{1}^i) + \mu_i(N) V_{t-1}(\mathbf{n}) + \theta_i (Q_i - n_i) V_{t-1}(\mathbf{n})\}}{A}, \\
&+ \max_u \left[\frac{\sum_{i=1}^M \mu_i(u_i) I_i \{r_i + V_{t-1}(\mathbf{n}-\mathbf{1}^i) - V_{t-1}(\mathbf{n})\}}{A} \right],
\end{aligned}
$$

where

$$
I_i = \begin{cases} 1 & , n_i > 0 \\ 0 & , n_i = 0 \end{cases},
$$

is an indicator function.

The first three terms in (4.3.1) account for an arrival to the system, a job completion and a job loss from the system, respectively. The last two terms include additional terms related to the null events which were introduced by the uniformization.

The indicator, I_i, is introduced to differentiate between the cases when there are jobs of class i to be processed and when none are present. If at least one job is present

in class i, then its service may be successfully completed and it may leave the system. However, if $n_i = 0$ then, clearly, no reward can be earned from completing a job of this class, neither can the number of jobs of class i be reduced so the state of the system remains unchanged in this respect. It is also true that no jobs of class i can be lost from the system.

Having presented the value iteration equations, we now describe how they can be applied to a problem in order to obtain the optimal solution.

Starting with $t = 1$, we calculate $V_t(\mathbf{n})$ for each vector $\mathbf{n}$, recording the service vector which achieves the maximum. This is repeated for all vectors $\mathbf{n}$ and then the quantities

$$m_t = \min_{\mathbf{n} \in X_{i=1}^{M}[0,Q_i]} (V_t(\mathbf{n}) - V_{t-1}(\mathbf{n})),$$
$$M_t = \max_{\mathbf{n} \in X_{i=1}^{M}[0,Q_i]} (V_t(\mathbf{n}) - V_{t-1}(\mathbf{n})),$$

are calculated.

We repeat the calculations for increasing values of t until

$$0 \leq M_t - m_t \leq \varepsilon,$$

where ε (very small, around 1×10^{-6}) has been chosen to give us our desired level of precision in our reward rate estimate. Then we can obtain our estimate of average reward rate, g, for the optimal policy from the identity:

$$\lim_{t \to \infty} [V_t(\mathbf{n}) - V_{t-1}(\mathbf{n})] = \frac{g}{A} \cong \frac{1}{2A}(M_T + m_T),$$

where T is the point at which the above iteration scheme stops.

The above scheme can be implemented to obtain an optimal policy for allocating the N servers between the M job classes. It also provides the value of the reward rate corresponding to any optimal policy.

4.3.1 Allocation Examples

Tables $4.3.1(a), 4.3.1(b), 4.3.1(c)$ and $4.3.1(d)$ show some examples of the service vectors produced by the dynamic programming algorithm for a range of problems for which $M{=}2$ and where $N{=}10$ servers are available to be allocated between these classes.

For jobs in example 1, $\boldsymbol{\lambda} = (1,1)$, $\boldsymbol{\theta} = (0.1, 0.1)$ and $\mathbf{r} = (1,1)$; for jobs in example 2, $\boldsymbol{\lambda}{=}(1, 1.5), \boldsymbol{\theta}{=}(0.1, 0.1)$ and $\mathbf{r}{=}(1,1)$; for jobs in example 3, $\boldsymbol{\lambda}{=}(1,1)$, $\boldsymbol{\theta} = (0.1, 0.2)$ and $\mathbf{r} = (1,1)$; for jobs in example 4, $\boldsymbol{\lambda} = (1,1), \boldsymbol{\theta} = (0.1, 0.1)$ and $\mathbf{r} = (1, 1.5)$. The service function used was $\mu_j(u) = log_e(u+1), 1 \leq j \leq 2$, for all examples. Thus comparison of results for examples $2-4$ can give some indication of how increasing the arrival rate, loss rate or reward respectively for one class can impact the service allocation policy. The numbers within the table are the optimum number of servers allocated to class 1 for the indicated state.

In all the examples, for each job class j, the number of servers allocated to serve that class increases with the queue length n_j. For example, in Table 4.3.1(a) when the state is $(1,1)$, 5 servers are allocated to class 1 (and so 5 servers allocated to class 2) while for state $(2,1)$ (one more job of class 1 is present) 6 servers are now allocated to class 1 (and so 4 servers are allocated to class 2). This follows our intuition for the problem - as more jobs are present in one class, we would expect more service effort to be directed towards jobs in that class. The fact that there are more jobs to be processed means this class would yield a greater expected return.

Some evidence is also provided to suggest that more service should be allocated to the job class with the higher arrival rate. For the $(3,4)$ state, the allocation of servers to class 1 falls from 5 in Table 4.3.1(a) to 4 in Table 4.3.1(b) which has a greater arrival rate for jobs of class 2. If more jobs of class 2 are arriving then it would seem reasonable to allocate more servers to this class.

Comparing the results for examples 1 and 3, suggests that more service is allocated

to the job class with higher loss rate. For the $(5, 1)$ state, the allocation of servers to class 1 falls from 7 in Table 4.3.1(a) to 6 in Table 4.3.1(c), the latter corresponding to a greater loss rate for jobs of class 2. This would seem reasonable since the service of jobs of class 2 is more urgent. Allocating more service to this class, thus providing quicker service, could mean jobs are completed before being lost from the system.

Finally, a comparison of examples 1 and 4 suggests that more service is also allocated to the job class with higher reward rate. For the $(1, 7)$ state, the allocation of servers to class 1 falls from 3 in Table 4.3.1(a) to 2 in Table 4.3.1(d), the latter corresponding to a larger reward associated with jobs of class 2. This again would follow intuition with service being directed in favour of the job class for which each service completion yields a greater reward.

While the dynamic programming algorithm described does deliver the optimal solution, the size of the state space is $\Pi_{i=1}^{M}(Q_i + 1)$. This greatly increases with the number of job classes, M, and prohibits the use of DP for problems with more than 2 or 3 job classes. This motivates our search for a strongly performing heuristic which can be computed more easily and for problems with large values of M. We seek a heuristic which allocates service following the same pattern/trends as our dynamic programming algorithm.

n_2 \ n_1	0	1	2	3	4	5	6	7	8	9	10
0	10	10	10	10	10	10	10	10	10	10	10
1	0	5	6	6	7	7	7	7	7	7	7
2	0	4	5	5	6	6	6	6	6	6	7
3	0	4	5	5	5	6	6	6	6	6	6
4	0	3	4	5	5	5	5	6	6	6	6
5	0	3	4	4	5	5	5	5	5	5	6
6	0	3	4	4	5	5	5	5	5	5	5
7	0	3	4	4	4	5	5	5	5	5	5
8	0	3	4	4	4	5	5	5	5	5	5
9	0	3	4	4	4	5	5	5	5	5	5
10	0	3	3	4	4	4	5	5	5	5	5

Table 4.3.1(a) Number of servers allocated to class 1 by the optimal schedule for state (n_1, n_2) for some two class problems of type 1.

n_2 \ n_1	0	1	2	3	4	5	6	7	8	9	10
0	10	10	10	10	10	10	10	10	10	10	10
1	0	5	6	6	6	6	7	7	7	7	7
2	0	4	5	5	5	6	6	6	6	6	6
3	0	3	4	5	5	5	5	5	6	6	6
4	0	3	4	4	5	5	5	5	5	5	5
5	0	3	4	4	4	5	5	5	5	5	5
6	0	3	4	4	4	5	5	5	5	5	5
7	0	3	3	4	4	4	5	5	5	5	5
8	0	3	3	4	4	4	4	5	5	5	5
9	0	3	3	4	4	4	4	5	5	5	5
10	0	3	3	4	4	4	4	4	5	5	5

Table 4.3.1(b) Number of servers allocated to class 1 by the optimal schedule for state (n_1, n_2) for some two class problems of type 2.

n_2 \ n_1	0	1	2	3	4	5	6	7	8	9	10
0	10	10	10	10	10	10	10	10	10	10	10
1	0	4	5	5	6	6	6	6	6	6	7
2	0	3	4	4	5	5	5	5	5	6	6
3	0	3	4	4	4	5	5	5	5	5	5
4	0	3	3	4	4	4	4	5	5	5	5
5	0	2	3	4	4	4	4	4	5	5	5
6	0	2	3	3	4	4	4	4	4	4	5
7	0	2	3	3	4	4	4	4	4	4	4
8	0	2	3	3	4	4	4	4	4	4	4
9	0	2	3	3	4	4	4	4	4	4	4
10	0	2	3	3	4	4	4	4	4	4	4

Table 4.3.1(c) Number of servers allocated to class 1 by the optimal schedule for state (n_1, n_2) for some two class problems of type 3.

n_2 \ n_1	0	1	2	3	4	5	6	7	8	9	10
0	10	10	10	10	10	10	10	10	10	10	10
1	0	4	5	6	6	6	6	6	7	7	7
2	0	3	4	5	5	5	5	6	6	6	6
3	0	3	4	4	4	5	5	5	5	5	5
4	0	3	3	4	4	4	5	5	5	5	5
5	0	3	3	4	4	4	4	4	4	5	5
6	0	2	3	3	4	4	4	4	4	4	4
7	0	2	3	3	4	4	4	4	4	4	4
8	0	2	3	3	4	4	4	4	4	4	4
9	0	2	3	3	3	4	4	4	4	4	4
10	0	2	3	3	3	4	4	4	4	4	4

Table 4.3.1(d) Number of servers allocated to class 1 by the optimal schedule for state (n_1, n_2) for some two class problems of type 4.

4.4 Static Policy

Our first step in developing a heuristic for this model is to develop an optimal static policy. A static policy allocates a proportion of servers to each job class, with this proportion remaining constant for all time. Therefore, this policy does not respond to the changing number of jobs in each class present at any time.

Under a static policy, the service rate for each job class is constant and the system evolves as a birth-death process. In this birth-death process, under the static policy corresponding to the fixed allocation $\mathbf{u}$ the birth/arrival rate for class j is λ_j and the state-dependent death/service completion rate is $\mu_j(u_j) + n\theta_j$ when class j is in state $n \in \mathbb{Z}^+$. Thus, from standard theory (see, for example Puterman (1994) or Mitrani (1998)) the long run proportion of time each job class j spends in any state n will converge to some limit. The collection of such limits gives the equilibrium distribution of the number of class j jobs in the system. It is given by

$$\Pi_n^j(\mu(u_j)) = \frac{\lambda_j^n \Pi_0(\mu(u_j))}{\prod_{m=1}^{n}(\mu_j(u_j)+m\theta_j)},$$

where

$$\Pi_0^j(\mu(u_j)) = \left[\sum_{n=0}^{\infty} \frac{\lambda_j^n}{\prod_{m=1}^{n}(\mu_j(u_j)+m\theta_j)}\right]^{-1}.$$

We can also interpret $\Pi_n^j(u_j)$ as the steady state probability that there are n jobs of class j in the system under the static policy which allocates u_j servers to class j. Our interpretation of $\Pi_0^j(u_j)$ is therefore as the steady state probability that the queue for class j is empty under the above policy.

The reward rate under $\mathbf{u}$ for each job class can simply be calculated by multiplying the successful completion rate under the static policy, by the probability that a job of that class is present to be processed, and also the reward earned from each successful completion. Thus, the average reward rate, $R(\mathbf{u})$, over the whole system under static policy $\mathbf{u}$ is given by:

$$R(\mathbf{u}) = \sum_{j=1}^{M} r_j \mu_j(u_j)[1 - \Pi_0^j(u_j)].$$

Our aim is to determine the static policy which maximises the above average reward rate over all job classes, subject to the constraint on the total number of servers. We write $\mathbf{u}^*$ for the maximising static policy, i.e.

$$R(\mathbf{u}^*) = \max\left\{R(\mathbf{u}); u_j \in \mathbb{Z}^+, 1 \leq j \leq M, \text{ and } \sum_{j=1}^{M} u_j = N\right\}. \tag{4.4.1}$$

We call any such $\mathbf{u}^*$ an optimal static policy.

Recall that $\mathbf{u}$ is a vector allocating a set of N servers between M job classes, where both N and M are finite, integer quantities. There are, therefore, a finite number of choices for $\mathbf{u}$ for any given problem, and so a maximising vector, $\mathbf{u}^*$ is guaranteed to exist.

4.5 Multiclass Improvement Policy

We now introduce a dynamic heuristic which is developed in a similar way to the heuristics in Chapter 3 and also generalises the analysis in Glazebrook et al. (2004). This heuristic is developed following a two stage procedure which takes the determination of the optimal static policy described in Section 4.4 as its first stage. In all states $\mathbf{n}$ we determine an optimal action given that the optimal static policy will be used at all future decision epochs. This constitutes a single DP policy improvement step applied to the static policy, $\mathbf{u}^*$, such that the action, $\mathbf{v}(\mathbf{n})$, chosen at the first decision epoch achieves a maximum of the total reward over all decision epochs in a suitable sense.

The resultant action $\mathbf{v}(\mathbf{n})$ is applied to the system at each decision epoch when the system state is $\mathbf{n}$, and so the static service vector $\mathbf{u}^*$ is never actually applied, only being deployed in the development of the multiclass improvement policy . Therefore, we can relax the condition that u_i must be integer. Thus, the first stage calculation determines $\mathbf{u}^*$ from (4.4.1), where, for $1 \leq j \leq M$, $u_j \in \mathbb{R}^+$. Such relaxation is guaranteed to yield a static policy for which $R(\mathbf{u})$ is at least as large as for the integer static policy, as integer solutions are not excluded. Thus, the relaxation improves the reward rate yielded by our multiclass improvement policy.

We will now describe the development of the 'multiclass improvement policy', but we first define the following quantities which will be required in order to develop and implement the procedure:

Definitions

$V(n_j, u_j^*, T)$ is the total expected reward earned by job class j in the period $[0, T)$ when being served by u_j^* servers and when there are n_j class-j jobs in the system at time 0.

$\hat{V}(n_j, u_j^*)$ is the total expected reward earned by job class j from the situation where there are n_j class-j jobs in the system at time 0 until the first transition into

the state $n_j = 0$ when being served by u_j^* servers.

$\hat{T}(n_j, u_j^*)$ is the expected time until the first transition into the state $n_j = 0$, for jobs in class j, when being served by u_j^* servers.

$b_j(n_j, u_j^*)$ is the bias function, or the difference in expected reward earned when there are n_j class-j jobs in the system rather than 0 jobs at time 0 when being served by u_j^* servers. Formally, we have that

$$b_j(n_j, u_j^*) = \lim_{T \to \infty} \left\{ V_j(n_j, u_j^*, T) - V_j(0, u_j^*, T) \right\}.$$

The following result is a simple consequence of implementing a single policy improvement step on the static policy $\mathbf{u}^*$, as described above.

Lemma 4.5.1. *The policy resulting from imposing a single policy improvement step upon the static policy* $\mathbf{u}^*$ *when in state* $\mathbf{n}$ *chooses the service vector* $\mathbf{v}^*$ *which maximises*

$$\sum_{j=1}^{M} r_j \mu_j(v_j) - \sum_{j=1}^{M} \mu_j(v_j) \left\{ b_j(n_j, u_j^*) - b_j(n_j - 1, u_j^*) \right\}.$$

subject to the constraints $\sum_{j=1}^{M} v_j = N, v_j \in \mathbb{N}, 1 \leq j \leq M$.

Proof. Recall the value iteration method and equation presented in Section 4.3. We modify equation (4.3.1) to reflect the fact that policy $\mathbf{u}^*$ is applied at all decisions after the first decision epoch. Thus, the maximisation associated with the policy improvement step is:

$$\begin{aligned} \max_{\mathbf{v}} \Bigg\{ & \frac{(\sum_{i=1}^{M} \lambda_i V_{t-1}(\mathbf{n}+\mathbf{1}^i, \mathbf{u}^*)}{A} + \frac{\sum_{i=1}^{M} \mu_i(v_i) I_i \{ r_i + V_{t-1}(\mathbf{n}-\mathbf{1}^i, \mathbf{u}^*) \}}{A} \\ & + \frac{\sum_{i=1}^{M} \theta_i n_i V_{t-1}(\mathbf{n}-\mathbf{1}^i, \mathbf{u}^*)}{A} + \frac{\sum_{i=1}^{M} \{ \mu_i(N) - \mu_i(v_i) I_i \} V_{t-1}(\mathbf{n}, \mathbf{u}^*)}{A} \\ & + \frac{\sum_{i=1}^{M} \theta_i (Q_i - n_i) V_{t-1}(\mathbf{n}, \mathbf{u}^*)}{A} \Bigg\} \end{aligned} \tag{4.5.1}$$

However, we note that only two of the terms in (4.5.1) are affected by the choice of $\mathbf{v}$ and thus the above maximisation is equivalent to maximising, with respect to $\mathbf{v}$,

the following:

$$\sum_{j=1}^{M} r_j \mu_j(v_j) - \sum_{j=1}^{M} \mu_j(v_j) \left\{ V_{t-1}(\mathbf{n}, \mathbf{u}^*) - V_{t-1}(\mathbf{n} - 1^j, \mathbf{u}^*) \right\}.$$

As the processes of arrivals and losses for distinct job classes are independent and the static policy is applied for all decisions after the initial decision epoch, the evolution of the system can be considered as independent queueing processes for each job class and so taking the limit $t \to \infty$ the quantity to be maximised becomes

$$\sum_{j=1}^{M} r_j \mu_j(v_j) - \sum_{j=1}^{M} \mu_j(v_j) \left\{ b_j(n_j, u_j^*) - b_j(n_j - 1, u_j^*) \right\}.$$

The quantity $b_j(n_j, u_j^*) - b_j(n_j - 1, u_j^*)$ represents the difference in expected reward earned from job class j which results from starting in state n_j rather than starting in state $n_j - 1$, under the static policy's allocation of service, u_j^*, over an infinite horizon. This is the form required. □

In order to implement the policy improvement step we need to evaluate the quantity $b_j(n_j, u_j^*) - b_j(n_j - 1, u_j^*)$ found in Lemma 4.5.1. By exploiting the fact that the system under static policy $\mathbf{u}^*$ evolves as a collection of independent birth-death processes, and applying results from renewal theory, the following Lemma yields a closed form for this quantity.

Lemma 4.5.2. *The quantity $b_j(n_j, u_j^*) - b_j(n_j - 1, u_j^*)$ can be evaluated as*

$$b_j(n_j, u_j^*) - b_j(n_j - 1, u_j^*) = \frac{r_j \mu_j(u_j^*)}{\{\mu_j(u_j^*) + \theta_j n_j\}} \frac{\left[\sum_{n=0}^{\infty} \lambda_j^n \left[\prod_{m=1}^{n} (\mu_j(u_j^*) + (m+n)\theta_j) \right]^{-1} \right]}{\left[\sum_{n=0}^{\infty} \lambda_j^n \left[\prod_{m=1}^{n} (\mu_j(u_j^*) + m\theta_j) \right]^{-1} \right]}.$$

Proof. Consider the queue length process $\{N_j(t), t \in \mathbb{R}^+\}$ under the static allocation u_j^*. This process evolves as a birth-death process with birth rate λ_j and death rate $\mu_j u_j^* + n\theta_j$ when in state $n \in \mathbb{Z}^+$.

According to standard MDP theory (see for example Puterman (1994) or Tijms (1994)), the birth-death process 'regenerates' at each entry to the empty state. This

regeneration property means that the behaviour of the process is probabilistically identical from each time the process enters the empty state (i.e., there are no jobs of class j present). Hence, the evolution of the process over an infinite horizon can be split into a number of independent cycles, where each cycle is the evolution of the process between two consecutive regeneration points. We can now consider a single cycle and how this relates to the full evolution of the process over an infinite horizon.

By results from MDP theory and the fact that entry into state 0 is a regeneration point for the process, we have that for all $n_j \geq 1$,

$$\begin{aligned} b_j(n_j, u_j^*) - b_j(n_j - 1, u_j^*) = &\hat{V}_j(n_j, u_j^*) - \hat{V}_j(n_j - 1, u_j^*) \\ &- r_j \mu_j(u_j^*)[1 - \Pi_0^j(\mu_j(u_j^*))]\{\hat{T}_j(n_j, u_j^*) - \hat{T}_j(n_j - 1, u_j^*)\}, \end{aligned} \tag{4.5.2}$$

Thus, our quantity over the infinite horizon is expressed in terms of the behaviour in one regeneration cycle. It remains to obtain expressions for $\hat{V}_j(n_j, u_j^*)$ and $\hat{T}_j(n_j, u_j^*)$.

In order to do this, we now consider the birth-death process $\{\check{N}(t), t \in \mathbb{R}^+\}$ starting at time zero with $\check{N}(0) = n_j$. This process is deemed to have birth rate λ_j and death rate $\mu_j(u_j^*) + (n_j + m)\theta_j$ when in states $(n_j + m), m \in \mathbb{N}$. While in state $(n_j - 1)$ or below, the death rate is zero, with interpretation including a withdrawal of service in these states. The transition rates for the process $\{\check{N}(t), t \in \mathbb{R}^+\}$ are therefore identical to those for $\{N(t), t \in \mathbb{R}^+\}$ for all states $m \geq n_j$. However, the process $\{\check{N}(t), t \in \mathbb{R}^+\}$ regenerates at each entry to the state $n_j - 1$ and enters no states below this.

Now, using the fact that the rewards earned during busy periods of $\{\check{N}(t), t \in \mathbb{R}^+\}$ form a sequence of i.i.d. random variables with mean $\hat{V}_j(n_j, u_j^*) - \hat{V}_j(n_j - 1, u_j^*)$, and that the process regenerates upon each entry into state $n_j - 1$, the average reward per unit time for the process is given by

$$\frac{\hat{V}_j(n_j, u_j^*) - \hat{V}_j(n_j - 1, u_j^*)}{\lambda^{-1} + \{\hat{T}_j(n_j, u_j^*) - \hat{T}_j(n_j - 1, u_j^*)\}}. \tag{4.5.3}$$

Moreover, by standard results from queueing theory for birth-death models, the

equilibrium distribution for $\check{N}(t)$ is given by

$$\check{\Pi}_n^j(\mu_j(u_j^*))=\begin{cases}\lambda_j^{n-n_j+1}\Pi_0^j(\mu_j(u_j^*)+(n_j-1)\theta_j)\left\{\prod_{m=n_j}^{n}(\mu_j(u_j^*)+m\theta_j)\right\}^{-1}, & n\geq n_j-1,\\ 0 & n\leq n_j-2,\end{cases}$$

where

$$\Pi_0^j(\mu_j(u_j^*)+(n_j-1)\theta_j)=\left[\sum_{n=0}^{\infty}\frac{\lambda_j^n}{\prod_{m=1}^{n}\left[\mu_j(u_j^*)+(n_j+m-1)\theta_j\right]}\right]^{-1}.$$

Therefore, the average reward rate per unit time earned during the evolution of $\check{N}(t)$ can also be expressed as

$$r_j\mu_j(u_j^*)[1-\Pi_{n_j-1}^j]=r_j\mu_j(u_j^*)[1-\Pi_0^j(\mu_j(u_j^*)+(n_j-1)\theta_j)]. \tag{4.5.4}$$

This is the reward earned for each successful service completion multiplied by the successful completion rate, multiplied by the probability that service is given to jobs.

Expressions (4.5.3) and (4.5.4) are both describing the same quantity, which leads us to conclude that

$$\frac{\hat{V}_j(n_j,u_j^*)-\hat{V}_j(n_j-1,u_j^*)}{\lambda^{-1}+\{\hat{T}_j(n_j,u_j^*)-\hat{T}_j(n_j-1,u_j^*)\}}=r_j\mu_j(u_j^*)[1-\Pi_0^j(\mu_j(u_j^*)+(n_j-1)\theta_j)]. \tag{4.5.5}$$

We have similarly that

$$\frac{\hat{T}_j(n_j,u_j^*)-\hat{T}_j(n_j-1,u_j^*)}{\lambda^{-1}+\{\hat{T}_j(n_j,u_j^*)-\hat{T}_j(n_j-1,u_j^*)\}}=1-\Pi_0^j(\mu_j(u_j^*)+(n_j-1)\theta_j). \tag{4.5.6}$$

Rearrangement of equation (4.5.6) now allows us to quantify $\hat{T}_j(n_j,u_j^*)-\hat{T}_j(n_j-1,u_j^*)$ as

$$\hat{T}_j(n_j,u_j^*)-\hat{T}_j(n_j-1,u_j^*)=\frac{\lambda^{-1}[1-\Pi_0^j(\mu_j(u_j^*)+(n_j-1)\theta_j)]}{\Pi_0^j(\mu_j(u_j^*)+(n_j-1)\theta_j)}, \tag{4.5.7}$$

and thus to quantify $\hat{V}_j(n_j,u_j^*)-\hat{V}_j(n_j-1,u_j^*)$ from equation (4.5.5). We now return to equation (4.5.2) to obtain our closed form for the difference in the total reward earned over an infinite horizon from initial states n_j and n_j-1 under static allocation

u_j^*. By substituting (4.5.5) and (4.5.7) into (4.5.2), we have

$$\begin{aligned}
b_j(n_j,u_j^*)-b_j(n_j-1,u_j^*) &= \hat{V}_j(n_j,u_j^*) - \hat{V}_j(n_j-1,u_j^*) \\
&\quad - r_j\mu_j(u_j^*)[1-\Pi_0^j(\mu_j(u_j^*))]\{\hat{T}_j(n_j,u_j^*) - \hat{T}_j(n_j-1,u_j^*)\}, \\
&= \frac{r_j\mu_j(u_j^*)[1-\Pi_0^j(\mu_j(u_j^*)+(n_j-1)\theta_j)]\lambda^{-1}}{\Pi_0^j(\mu_j(u_j^*)+(n_j-1)\theta_j)} \\
&\quad -\left\{r_j\mu_j(u_j^*)[1-\Pi_0^j(\mu_j(u_j^*))]\right\}\left\{\frac{\lambda^{-1}[1-\Pi_0^j(\mu_j(u_j^*)+(n_j-1)\theta_j)]}{\Pi_0^j(\mu_j(u_j^*)+(n_j-1)\theta_j)}\right\}, \\
&= \frac{r_j\mu_j(u_j^*)\Pi_0^j(\mu_j(u_j^*))}{(\mu_j(u_j^*)+n_j\theta_j)\Pi_0^j(\mu_j(u_j^*)+(n_j-1)\theta_j)}, \\
&= \frac{r_j\mu_j(u_j^*)}{\{\mu_j(u_j^*)+\theta_j n_j\}} \frac{\left[\sum_{n=0}^{\infty}\lambda_j^n\left[\prod_{m=1}^{n}(\mu_j(u_j^*)+(m+n)\theta_j)\right]^{-1}\right]}{\left[\sum_{n=0}^{\infty}\lambda_j^n\left[\prod_{m=1}^{n}(\mu_j(u_j^*)+m\theta_j)\right]^{-1}\right]},
\end{aligned}$$

which is the required form for $b_j(n_j,u_j^*)-b_j(n_j-1,u_j^*)$. □

We have now evaluated the difference in bias from starting in state n_j at time zero rather than in state n_j-1, and thus direct substitution of the result from Lemma 4.5.2 into Lemma 4.5.1 yields the following result.

Theorem 4.5.3. *The policy resulting from imposing a single policy improvement step upon the static policy* $\mathbf{u}^*$ *when in state* $\mathbf{n}$ *chooses the service vector* $\mathbf{v}^*$ *to maximise*

$$\sum_{j=1}^{M} r_j\mu_j(v_j)\left\{1-\frac{r_j\mu_j(u_j^*)}{\{\mu_j(u_j^*)+\theta_j n_j\}}\frac{\left[\sum_{n=0}^{\infty}\lambda_j^n\left[\prod_{m=1}^{n}(\mu_j(u_j^*)+(m+n)\theta_j)\right]^{-1}\right]}{\left[\sum_{n=0}^{\infty}\lambda_j^n\left[\prod_{m=1}^{n}(\mu_j(u_j^*)+m\theta_j)\right]^{-1}\right]}\right\}$$

subject to the constraints $\sum_{j=1}^{M} v_j = N, v_j \in \mathbb{N}, 1 \le j \le M$.

Theorem 4.5.3 presents a closed form for the multiclass improvement policy. At each decision epoch, this policy examines the state of the system, $\mathbf{n}$, and assigns the servers between job classes according to the associated $\mathbf{v}^*(\mathbf{n})$ as described in Theorem 4.5.3.

This policy can be computed much more easily than DP for the same problem size. The policy can also be implemented in problems with many job classes where

DP would be impractical, as its computation does not suffer the same problems as DP does with the size of the state space.

4.5.1 Allocation Examples

Tables 4.5.1(a), 4.5.1(b), 4.5.1(c) and 4.5.1(d) show the service vectors yielded by our heuristic policy determined by DP policy improvement for the same four examples as described in Section 4.3.1. That is, for jobs in example 1, $\boldsymbol{\lambda} = (1, 1)$, $\boldsymbol{\theta} = (0.1, 0.1)$ and $\mathbf{r} = (1, 1)$; for jobs in example 2, $\boldsymbol{\lambda} = (1, 1.5), \boldsymbol{\theta} = (0.1, 0.1)$ and $\mathbf{r} = (1, 1)$; for jobs in example 3, $\boldsymbol{\lambda} = (1, 1)$, $\boldsymbol{\theta} = (0.1, 0.2)$ and $\mathbf{r} = (1, 1)$; for jobs in example 4, $\boldsymbol{\lambda} = (1, 1), \boldsymbol{\theta} = (0.1, 0.1)$ and $\mathbf{r} = (1, 1.5)$. The decisions show the same general trends as for the DP algorithm described in Section 4.3, with more servers allocated to the class with more jobs. It appears that more servers are allocated to the job class with greater loss rate, or with larger arrival rate, or with larger reward. The multiclass improvement policy, however, is more conservative than DP, allocating servers more evenly between the two job classes. For example, in the $(1, 9)$ case for problem type 1, while there is only 1 job of class 1 present, the policy still allocates 3 servers to class one where the DP policy allocates only 1 server.

n_2 \ n_1	0	1	2	3	4	5	6	7	8	9	10
0	10	10	10	10	10	10	10	10	10	10	10
1	0	5	7	8	8	8	9	9	9	9	9
2	0	3	5	6	7	7	7	8	8	8	8
3	0	2	4	5	6	6	6	7	7	7	7
4	0	2	3	4	5	5	6	6	6	6	7
5	0	2	3	4	5	5	5	6	6	6	6
6	0	1	3	4	4	5	5	5	6	6	6
7	0	1	2	3	4	4	5	5	5	5	6
8	0	1	2	3	4	4	4	5	5	5	5
9	0	1	2	3	4	4	4	5	5	5	5
10	0	1	2	3	3	4	4	4	5	5	5

Table 4.5.1(a) Number of servers allocated to class 1 by the multiclass improvement policy for state (n_1, n_2) for some two class problems of type 1.

n_2 \ n_1	0	1	2	3	4	5	6	7	8	9	10
0	10	10	10	10	10	10	10	10	10	10	10
1	0	5	7	7	8	8	8	9	9	9	9
2	0	3	5	6	6	7	7	7	8	8	8
3	0	2	4	5	6	6	6	7	7	7	7
4	0	2	3	4	5	5	6	6	6	6	6
5	0	2	3	4	4	5	5	6	6	6	6
6	0	1	3	4	4	5	5	5	5	6	6
7	0	1	3	3	4	4	5	5	5	5	5
8	0	1	2	3	4	4	4	5	5	5	5
9	0	1	2	3	4	4	4	5	5	5	5
10	0	1	2	3	3	4	4	4	5	5	5

Table 4.5.1(b) Number of servers allocated to class 1 by the multiclass improvement policy for state (n_1, n_2) for some two class problems of type 2.

n_2 \ n_1	0	1	2	3	4	5	6	7	8	9	10
0	10	10	10	10	10	10	10	10	10	10	10
1	0	4	6	6	7	7	8	8	8	8	8
2	0	2	4	5	5	6	6	6	7	7	7
3	0	2	3	4	5	5	5	6	6	6	6
4	0	1	3	3	4	5	5	5	5	6	6
5	0	1	2	3	4	4	4	5	5	5	5
6	0	1	2	3	3	4	4	4	5	5	5
7	0	1	2	3	3	4	4	4	4	5	5
8	0	1	2	3	3	4	4	4	4	5	5
9	0	1	2	2	3	3	4	4	4	4	5
10	0	1	2	2	3	3	4	4	4	4	4

Table 4.5.1(c) Number of servers allocated to class 1 by the multiclass improvement policy for state (n_1, n_2) for some two class problems of type 3.

n_2 \ n_1	0	1	2	3	4	5	6	7	8	9	10
0	10	10	10	10	10	10	10	10	10	10	10
1	0	4	6	7	7	8	8	8	8	9	9
2	0	3	4	5	6	6	6	7	7	7	7
3	0	2	3	4	5	5	6	6	6	6	6
4	0	1	3	4	4	5	5	5	5	6	6
5	0	1	2	3	4	4	4	5	5	5	5
6	0	1	2	3	3	4	4	4	5	5	5
7	0	1	2	3	3	3	4	4	4	4	5
8	0	1	2	2	3	3	4	4	4	4	4
9	0	1	2	2	3	3	3	4	4	4	4
10	0	0	1	2	3	3	3	3	4	4	4

Table 4.5.1(d) Number of servers allocated to class 1 by the multiclass improvement policy for state (n_1, n_2) for some two class problems of type 4.

4.6 Numerical Results

Problems were studied for a range of cases for various arrival and loss rates as well as various values of reward available, all in the situation where $M=2$. As there are only two job classes, the DP policy can be calculated and the reward rate of our proposed policies measured against the optimal rate obtained from application of DP. Tables $4.6.1(a), 4.6.1(b)$ and $4.6.1(c)$ show estimates of the average reward rates earned under the policies studied:

- Static policy, $\mathbf{u}^*$
- multiclass improvement policy
- optimal policy derived by using DP value iteration.

The values for the static policy were given by expression (4.4.1). For the DP policy, values were obtained using the value iteration algorithm described in Section 4.3. Estimates of expected reward earned for the multiclass improvement policy were obtained using Monte Carlo simulation. Each example was simulated 200 times and

the average reward rate earned calculated. This procedure ensured the standard errors (shown in brackets) were sufficiently small to allow reward rate comparisons between the policies to be meaningful.

Construction of the Monte Carlo simulation was greatly simplified by exploiting the fact that arrival times, loss times and service completion times were all exponentially distributed and thus exhibit the memoryless property. Hence, at each event, the servers could be reallocated according to the policy and the time until the next event (whether it is an arrival, loss or service completion) resampled rather than keeping track of the whole system.

Both the static and multiclass improvement policies were also implemented using value iteration (4.3.1), with the service vector from the policy replacing $\mathbf{v}$ at each step. Agreement between the results gained from value iteration methods and equation (4.4.1) for the static policy, and simulation for the multiclass improvement policy provided reassurance that the results from each method were accurate. The results from utilising value iteration for the improvement policy are included for completeness.

Upon examination of the general characteristics of the results produced, Table 4.6.1(a) shows that as the loss rate for one job class increases, the total reward rate decreases, as would be expected. If jobs of one class are more likely to leave the system then there will be fewer jobs to process and a smaller reward earned.

When the arrival rate of class 2 jobs increases, the reward rate also increases, as the total number of jobs in the system is greater, so there is a larger potential reward to be earned. As would be expected, more service effort is directed to the job class with the larger arrival rate.

Now, comparing the proposed policies, we see that the multiclass improvement policy yields greater rewards than those earned from the static policy. This is indeed guaranteed from the method of developing the improvement policy. However, the

multiclass improvement policy only yields 2–6% greater reward than the static policy across all examples, with the greatest difference between the policies being when one class of jobs has a large loss rate.

A comparison of the multiclass improvement policy with the optimal (DP) policy shows that the multiclass improvement policy is very close to optimal, yielding less than 0.1% difference in reward rate in most examples. Thus, while our heuristic may not make large gains on the static policy, this appears to be because the static policy actually yields rewards which are relatively close to optimal rather than because the multiclass improvement policy does not perform well.

θ_2	$\mathbf{u}^*$	$\mathbf{u}^*$ improv. simulation	$\mathbf{u}^*$ improv. iteration	DP
0.1	1.8107	1.8468 (0.0027)	1.8490	1.8502
0.2	1.7468	1.7929 (0.0027)	1.7950	1.7959
0.5	1.6195	1.6813 (0.0026)	1.6817	1.6826
0.8	1.5340	1.6015 (0.0025)	1.6040	1.6042
1	1.4892	1.5551 (0.0023)	1.5623	1.5624
1.2	1.4511	1.5200 (0.0024)	1.5260	1.5265
1.5	1.4031	1.4721 (0.0025)	1.4807	1.4808
2	1.3402	1.4141 (0.0024)	1.4198	1.4200

Table 4.6.1(a) Estimates of the reward rate earned under three policies, for problems with various values of θ_2, where $\theta_1 = 0.1$, $\boldsymbol{\lambda} = (1, 1)$, $\boldsymbol{\nu} = (log_e(u+1), log_e(u+1))$ and $\mathbf{r} = (1, 1)$.

r_2	$\mathbf{u}^*$	$\mathbf{u}^*$ improv. simulation	$\mathbf{u}^*$ improv. iteration	DP
0.5	1.3608	1.3899 (0.0023)	1.3882	1.3888
0.8	1.6300	1.6649 (0.0025)	1.6646	1.6652
1	1.8107	1.8468 (0.0027)	1.8490	1.8502
1.5	2.2650	2.3090 (0.0038)	2.3125	2.3134
2	2.7215	2.7741 (0.0041)	2.7764	2.7776
3	3.6384	3.7112 (0.0061)	3.7064	3.7079
4	4.5584	4.6394 (0.0087)	4.6377	4.6394

Table 4.6.1(b) Estimates of the reward rate earned under three policies, for problems with various values of r_2, where $r_1=1$, $\boldsymbol{\nu}=(log_e(u+1), log_e(u+1))$, $\boldsymbol{\theta}=(0.1, 0.1)$, and $\boldsymbol{\lambda}=(1, 1)$.

λ_2	$\mathbf{u}^*$	$\mathbf{u}^*$ improv. simulation	$\mathbf{u}^*$ improv. iteration	DP
0.2	1.1105	1.1257 (0.0024)	1.1256	1.1258
0.5	1.3781	1.4026 (0.0023)	1.4032	1.4034
1	1.8107	1.8468 (0.0027)	1.8490	1.8502
1.5	2.2101	2.2594 (0.0032)	2.2609	2.2629
2	2.5507	2.6144 (0.0028)	2.6114	2.6151
3	2.9367	3.0005 (0.0032)	2.9990	3.0069

Table 4.6.1(c) Estimates of the reward rate earned under three policies, for problems with various values of λ_2, where $\lambda_1=1$, $\boldsymbol{\nu}=(log_e(u+1), log_e(u+1))$, $\boldsymbol{\theta}=(0.1, 0.1)$ and $\mathbf{r}=(1, 1)$.

4.7 Summary

In this chapter we studied a multiclass queueing system in which each job arrives at a central processing point and is prepared to wait an unknown length of time before leaving the system. Each job belongs to one of a number of classes and service is available in the form of a group of servers who could be divided among job classes.

A dynamic programming algorithm was formulated and two heuristics were developed. One heuristic was static, allocating a constant proportion of servers to each job class, while the other heuristic dynamically allocated servers according to the number of servers of each job class in the system. The dynamic heuristic was developed using the idea that at all future epochs after the first decision epoch, a constant proportion of servers would be allocated between each job class, with this proportion determined by the static heuristic.

Numerical examples showed that both the dynamic heuristic and DP algorithm allocated more servers to the job class with more jobs present. They also allocated more servers to the job class with the greater loss rate, greater arrival rate or larger reward available (with all other parameters equal). In numerical study the static heuristic yielded rewards within 6% of the optimal solution while the dynamic heuristic performed better, yielding rewards within 0.1% of the optimal solution. This suggested that both heuristics perform well in comparison to the optimal solution obtained from DP. Both heuristics were also more computationally efficient than the DP algorithm, indicating that they could be practical methods for solving this type of problem.

Chapter 5

Conclusion and Further Research

In this thesis, we examined decision problems concerned with the scheduling of jobs for which successful service completion can not immediately be seen or is uncertain. This is an area which has so far gone widely unstudied. A range of problems were investigated. In Chapters 2 and 3, a server sought to allocate a fixed time to each job, with the length of time depending on the number of jobs present. In Chapter 4, a central controller allocated a group of servers between a number of classes of jobs, again depending on the number of jobs present. In all cases the allocation of more service to one job/class may have made successful completion more likely, but would impose a burden on other jobs/classes. Such problems can be formulated using DP methods, however DP is only practical for problems of small size. Therefore, the purpose of this thesis was to develop methods for dynamically allocating service in order to maximise expected reward, which could be used as alternatives to DP. Optimal schedules were characterised and heuristics developed and evaluated.

In Chapter 2, models were examined where a single server was presented with a collection of jobs to be processed. These jobs were available indefinitely but a discount factor was applied to rewards earned, and thus early processing was encouraged. A calibrating index (the Gittins index) was developed which could be calculated based

only on information about the job under consideration, independently of other jobs that may also be present. This index is easily computable and was shown to define a set of times for each job (minimal times) of which any time chosen in an optimal schedule must be a member. The power of the Gittins Index was shown when all the jobs to be processed were identical. Here the index defined the total expected reward available from a large number of jobs, and also defined an optimality range, the range of jobs for which each minimal time is optimal. This greatly simplified the task of finding the optimal schedule when all jobs are identical.

The problem was also modelled as a restless bandit, and two heuristics based on Whittle indices were produced, both easily computable. The Whittle index developed is closely related to the Gittins Index and could again be calculated using only information about a single job. Both of the heuristics based on the Whittle index were evaluated by comparing their performance to that of the optimal solution (gained using DP) in 400 randomly generated problems. Both were found to yield results within 0.2% of the optimal solution. The 'improved Whittle heuristic' was found to yield results very close to optimal (within 0.07% of optimality).

In Chapter 3, a service system was examined in which jobs arrived at a single server and were prepared to wait an unknown length of time before being lost from the system. Previous research on this model had concentrated on developing and evaluating static policies for service and dynamic policies had not been considered. Hence, the development of dynamic policies for this model was an outstanding issue. A dynamic programming algorithm was defined and four alternative heuristics developed, all allocating a fixed time to each job. Two of these heuristics were relatively simple, with one allocating exponential times to each job, while another allocated a constant time. The other two heuristics dynamically allocated processing times depending on how many jobs were currently present, imagining that all future jobs will be allocated some constant service as prescribed

by one of the simpler heuristics described above. The dynamic heuristics were more easily computed than DP and the best performing one was found to yield results within 1% of the solutions obtained from DP.

In Chapter 4, a multiclass queueing system was studied. In this system, each job arrived at a central processing point and was prepared to wait an unknown length of time before leaving the system. Each job belonged to one of a number of classes and service was available in the form of a group of servers who could be divided among job classes. Two heuristics were developed and evaluated against the optimal solution obtained from DP methods. One heuristic was static, allocating a constant proportion of servers to each job class, while the other heuristic allocated servers dynamically. This dynamic allocation of servers also used the idea that at all future decisions a constant proportion of servers would be allocated between each job class, with the proportion determined by the static heuristic. Work in this chapter sought to generalise previous research on a single server version of this model to the multiple server case.

The heuristics developed in this research were all more easily computable than their respective DP equivalents. This was more pronounced in Chapter 3 where the computation time was reduced from more than 2 hours for DP to a matter of seconds for the heuristics. While being more computationally efficient than DP, the heuristics developed were also shown to yield rewards close to those obtained from DP. In Chapter 2 the improved Whittle heuristic yielded results within 0.1% of the optimal solution, while in Chapter 3 the $\frac{1}{\mu^*}$ improvement policy yielded results within 1% of the optimal solution and in Chapter 4 the multiclass improvement policy yielded results within 0.1% of the optimal solution. This provides indication that the heuristics developed could be practical alternatives to DP for the classes of problems studied. This is of particular interest for the multiclass problem in Chapter 4 (and potentially the multiclass problem in Chapter 2), where DP is not practical

for problems with a large number of classes (i.e., large M). Therefore, in cases where M is large, heuristics may be the only way to gain a solution for the problem, rather than merely providing an easier method.

For the problems studied in Chapters 3 and 4, the static policies, in which service is allocated identically irrespective of the number of jobs in the system, were generally found to yield rewards within 5% of the optimal dynamic solutions. It was rather surprising to find that static policies performed so well in comparison to the dynamic class of policies. This strong performance may be related to the service functions used in the models, or may be a characteristic of the problems considered themselves. It would therefore be interesting to examine the relative performance of the proposed static and dynamic policies in a wider range of problems to determine if there are situations in which a server would be content to implement a static policy so that they don't need to waste effort switching service between jobs/classes. Other areas of research also include modifying the models to include start up and teardown costs when moving onto another job, e.g., to reflect delays associated with changing jobs or moving servers between job classes. Incorporating these costs into the model should discourage switching between jobs or job classes and so should further enhance the performance of the static policies.

The performance of the heuristics proposed in this study has been evaluated numerically. The development of formal methods for evaluating the performance of these heuristics, e.g., by deriving bounds on their performance, would be very challenging, but would allow the heuristics to be used with greater confidence. The numerical results from this study do, however, indicate that these heuristics are practical methods for deriving near optimal policies for scheduling jobs of uncertain duration and uncertain or unobservable outcome. Further research could also seek to identify other situations to which our method of developing heuristics could be usefully applied.

Bibliography

Ansell, P. S., Glazebrook, K. D., Mitrani, I. and Niño Mora, J. (1999), A semidefinite programming approach to the optimal control of a single server queueing system with imposed second moment constraints. *J. Oper. Res. Soc.*, **50**, 765–773.

Banks, J. S. and Sundaram, R. K. (1994), Switching costs and the gittins index. *Econometrica*, **62**, 687–694.

Bassamboo, A., Harrison, J. M. and Zeevi, A. (2005), Dynamic routing and admission control in high-volume service systems: asymptotic analysis via multi-scale fluid limits. *Queueing Syst.*, **51**, 249–285.

Bellman, R. (1954), Dynamic programming and a new formalism in the calculus of variation. *Proc. of the Nat. Acad. of Sci. of the Unit. States of Amer.*, **40**.

Bellman, R. (1957), *Dynamic Programming*. Princeton University Press.

Benkherouf, L., Glazebrook, K. D. and Owen, R. W. (1991), Gittins indices and oil exploration. *J. Roy. Statist. Soc B*, **54**, 229–241.

Benkherouf, L., Glazebrook, K. D. and Owen, R. W. (1993), Single-visit policies for allocating a single resource in a stochastic environment. *Oper. Res.*, **42**, 1087–1099.

Bertsimas, D. and Niño Mora, J. (1996), Conservation laws, extended polymatroids and multi-arned bandit problems: a unified approach to indexable systems. *Maths. Oper. Res.*, **21**, 257–306.

Boots, N. K. and Tijms, H. (1999), A multi-server queueing system with impatient customers. *Mgmt. Sci.*, **45**, 444–448.

Dacre, M., Glazebrook, K. D. and Niño Mora, J. (1999), The achievable region approach to the optimal control of stochastic systems. *J. Roy. Statist. Soc B*, **61**, 747–791.

Doytchinov, B., Lehoczky, J. and Shreve, S. (2001), Real-time queues in heavy traffic with earliest-deadline-first queue discipline. *Ann. Appl. Prob.*, **2**, 332–378.

Garnett, O., Mandelbaum, A. and Reiman, M. (2002), Designing a call center with impatient customers. *Manuf. and Service Oper. Mgmt.*, **4**, 208–227.

Gaver, D. P., Jacobs, P. A., Samorodnitsky, G. and Glazebrook, K. D. (2006), Modeling and analysis of uncertain time-critical tasking problems. *Nav. Res. Logist.*, **53**, 588–599.

Gittins, J. C. (1979), Bandit processes and dynamic allocation indices. *J. Roy. Statist. Soc B*, **42**, 148–177.

Gittins, J. C. (1989), *Multi-armed bandit allocation indices.* Wiley, Chichester.

Gittins, J. C. and Jones, D. M. (1974), *Prog. in Statist.*, chap. A dynamic allocation index for the design of experiments. North Holland, Amsterdam.

Gittins, J. C. and Jones, D. M. (1979), A dynamic allocation index for the discounted multiarmed bandit problem. *Biometrika*, **66**, 561–565.

Glazebrook, K. D. (1980), On stochastic scheduling with precedence relations and switching costs. *J. Appl. Prob. Soc. B*, **17**, 1016–1024.

Glazebrook, K. D. (1983), Stochastic scheduling with due dates. *Int. J. Syst. Sci.*, **14**, 1259–1271.

Glazebrook, K. D., Ansell, P., Dunn, R. T. and Lumley, R. R. (2004), On the optimal allocation of service to impatient tasks. *J. Appl. Prob.*, **40**, 51–72.

Glazebrook, K. D., Niño Mora, J. and Ansell, P. (2002), Index policies for a class of discounted restless bandits. *Adv. Appl. Prob.*, **34**, 754–774.

Glazebrook, K. D. and Punton, E. L. (2005), Fixed-time schedules for the processing of jobs when service completions are not observable. *Meth. Math. Oper. Res.*, **62**, 77–97.

Glazebrook, K. D. and Punton, E. L. (2007), Dynamic policies for uncertain time-critical tasking problems. *forthcoming in Nav. Res. Logist.*

Harrison, J. M. and Zeevi, A. (2004), Dynamic scheduling of a multi-class queue in the halfin-whitt heavy traffic regime. *Oper. Res.*, **52**, 243–257.

Hillier, F. S. and Lieberman, G. J. (2005), *Introduction to Operations Research.* McGraw-Hill, New York.

Jiang, Z., Lewis, T. G. and Colin, J. Y. (1996), Scheduling hard real-time constrained periodic tasks on multiple processors. *J. Syst. Software*, **19**, 102–118.

Katehakis, M. N. and Veinott, A. F. (1987), The multiarmed bandit problem: Decomposition and computation. *Math. Oper. Res.*, **12**, 262–268.

Lehoczky, J. P. (1997), Using real-time queueing theory to control lateness in real-time systems. *Perform. Eval. Rev.*, **25**, 158–168.

Mitrani, I. (1998), *Probabilistic Modelling.* Cambridge University Press, Cambridge.

Nash, P. (1980), A generalized bandit problem. *J. Roy. Statist. Soc. B*, **42**, 165–169.

Niño Mora, J. (2001), Restless bandits, partial conservation laws and indexability. *Adv. Appl. Prob.*, **33**, 76–98.

Papadimitrou, C. H. and Tsitsiklis, J. N. (1999), The complexity of optimal queueing network control. *Math. Oper. Res.*, **24**, 293–305.

Puterman, M. L. (1994), *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* Wiley, New York.

Ross, S. M. (1983), *Introduction to Stochastic Dynamic Programming.* Academic Press Inc, New York.

Tijms, H. C. (1994), *Stochastic Models An Algorithmic Approach.* Wiley, Chichester.

Tsitsiklis, J. N. (1994), A short proof of the gittins index theorem. *Ann. Appl. Prob.*, **4**, 194–199.

Varaiya, P. P., Walrand, J. C. and Buyukkoc, C. (1985), Extensions of the multi-armed bandit problem: the discounted case. *IEEE Trans. AC.*, **30**, 426–439.

Weber, R. (1992), On the gittins index for multiarmed bandits. *Ann. Appl. Prob.*, **2**, 1024–1033.

Weber, R. and Weiss, G. (1990), On an index policy for restless bandits. *J. Appl. Prob.*, **27**, 637–648.

Weber, R. and Weiss, G. (1991), Addendum to on an index policy for restless bandits. *Adv. Appl. Prob.*, **23**, 637–648.

Weiss, G. (1988), Branching bandit processes. *Prob. Engng. Inform. Sci.*, **2**, 269–278.

White, D. (1993), *Markov decision processes.* Wiley, Chichester.

Whitt, W. (1999), Improving service by informing customers about anticipated delays. *Mgmt Sci.*, **45**, 192–207.

Whittle, P. (1980), Multi-armed bandits and the gittins index. *J. Roy. Statist. Soc. B*, **42**, 143–149.

Whittle, P. (1981), Arm-acquiring bandits. *Ann. Appl. Prob.*, **9**, 284–292.

Whittle, P. (1988), Restless bandits: activity allocation in a changing world. *J. Appl. Prob. Spec.*, **25A**, 287–298.

Appendices

Appendix A

The following is a paper published in Mathematical Methods of Operations Research in 2005. It contains material covered in Chapter 2 of the thesis.

Math. Meth. Oper. Res. (2005) 62: 77–97
DOI 10.1007/s00186-005-0442-5

ORIGINAL ARTICLE

K.D. Glazebrook · E.L. Punton

Fixed-time schedules for the processing of jobs when service completions are not observable

Received: November 2004 / Revised: January 2005
© Springer-Verlag 2005

Abstract We argue the importance of studying service systems in which the successful completion of job/customer service cannot be observed contemporaneously. Military and other applications are cited. The allocation of a large amount of processing to a job may make its (unobservable) successful completion more likely but will impose a burden on other jobs awaiting service. A fixed-time schedule specifies both the order in which jobs should be processed and how much processing each should receive. The goal is to find schedules which maximise the total expected reward earned from all jobs served. While this problem is intractable in general, a range of characterisations of optimal fixed-time schedules is achieved for given scenarios. The development of effective heuristics is also discussed.

Keywords Gittins index · restless bandit · stochastic dynamic programming · stochastic scheduling

1 Introduction

In most conventional models of service systems, including for example, queueing systems, it is assumed that jobs/customers are dispatched from the system once service is complete. A pervasive assumption (often unspoken) is that (successful) service completion is always observable and, indeed, is always observed, usually instantaneously. Gaver, Jacobs and Samorodnitsky (2003) point out that such assumptions may not hold good in many applications, including military ones. In a military setting, "service" may consist of shooting at a target. It may not be possible to establish in real time whether a target has been destroyed. More generally, a server's goal may be to neutralise a threat by searching an area and seeking to

In honour of the 60th birthday of Professor Ulrich Rieder

K.D. Glazebrook (✉), E.L. Punton
School of Management, University of Edinburgh, Edinburgh, EH8 9JY, UK.
E-mail: Kevin.Glazebrook@ed.ac.uk

destroy any opposing forces discovered. Such "service" may be partial or incomplete because of a deficiency of time, information or resources. However, it will tend to be true that the more time and resources are allocated to such service, the more likely it is to be completed successfully. The same may also be true , *inter alia*, in many situations in which the goal of service is some form of diagnosis (whether medical or otherwise) and/or repair.

We propose a simple class of models in which a single server has a collection of jobs to perform. A reward is earned upon successful completion of each job. While successful completion is not observable contemporaneously, it is more likely to be achieved if a greater amount of processing time is allocated to it. However, the latter imposes a burden on the jobs awaiting service. In our models this burden is mediated by a discounted reward structure. The latter might arise from assumptions about the (limited) lifespan of the service provider (which is also a salient feature of military applications). The system controller must decide *both* the order in which jobs should be processed *and* how much processing to allocate to each. The goal is to maximise the total expected reward earned from all jobs in a situation in which the actual rewards earned cannot be immediately known. Jobs which are high reward and which are likely to be completed quickly should feature early in the schedule but may receive little processing since early on more waiting jobs are penalised by any delay.

The problem of determining optimal *fixed-time schedules* is introduced in Section 2. That such problems are intractable in general is substantiated by a later discussion (in Section 5) which argues that they may be regarded as *restless bandit* problems. The latter is a class of decision problems introduced by Whittle (1988) and shown to be PSPACE-hard by Papadimitriou and Tsitsiklis (1999). Our problems also fall within the intractable class of *single visit* problems, introduced by Benkherouf, Glazebrook and Owen (1994) who discuss the development of heuristics, as does Whittle (1988). Notwithstanding this intractability, we succeed in developing a range of characterisations of optimal fixed-time schedules in Sections 2–4. The theoretical results presented there explore how the trade-offs inherent in fixed-time schedule construction are best resolved in a range of problem scenarios. The paper concludes in Section 5 with a discussion of heuristic development based primarily on restless bandit ideas.

Through the paper extensive use is made of reward rate notions related to the *Gittins index*. There is a substantial general literature on the index (see, for example, Gittins (1989), Glazebrook (1995) and Dacre, Glazebrook and Niño-Mora (1999) and references therein). However, our problems are dominated by special simplifying features and we have sought to exploit these to produce direct and transparent analyses. Our hope is that our discussion will be accessible to readers who are unfamiliar with index theory and that the power of reward rate ideas will be well evidenced by the material. We believe that these goals would be especially welcome to Professor Rieder who has a longstanding interest in bandit problems (see, for example, Rieder and Wagner (1991)) and in whose honour the paper is written.

2 Fixed-time schedules

The goal of analysis is the development of an optimal *fixed-time schedule* of a collection $J = \{1, 2, \ldots, N\}$ of jobs as follows:

(a) Each job $j \in J$ has a *processing requirement* P_j, a random variable taking values in the positive integers and with finite support $\{1, 2, \ldots, T_j\}$. We write

$$p_j(s) = P\{P_j = s\},\ 1 \le s \le T_j,\ j \in J.$$

Processing requirements for distinct jobs are supposed independent. Job j also has an associated positive *reward* r_j. Successful completion of job j at time t earns a reward $\beta^t r_j$, where $\beta \in (0, 1)$ is a *discount rate*. We write

$$R_j(t) = \sum_{s=1}^{t} \beta^s r_j p_j(s),\ 1 \le t \le T_j,\ j \in J, \tag{1}$$

for the expected reward achieved when job j is processed for t time units beginning at time zero.

(b) A *fixed-time schedule* for J is given by a pair $(\pi, \mathbf{t})$ where π is a permutation of J and $\mathbf{t}$ is an N-vector whose j^{th} component is an integer in the range $[1, T_{\pi_j}]$. We interpret $(\pi, \mathbf{t})$ as follows: at time 0, job π_1 is scheduled for processing until time t_{π_1}. Then job π_2 is scheduled for processing from t_{π_1} until time $t_{\pi_1} + t_{\pi_2}$, and so on. We write the total expected reward associated with $(\pi, \mathbf{t})$ as

$$V(\pi, \mathbf{t}) = \sum_{k=1}^{N} \beta^{\sum_{j=1}^{k-1} t_{\pi_j}} R_{\pi_k}(t_{\pi_k}). \tag{2}$$

The goal of analysis is to choose a pair $(\pi, \mathbf{t})$ to maximise $V(\pi, \mathbf{t})$.

(c) Complete enumeration of $V(\pi, \mathbf{t})$ for each fixed-time schedule involves $N!(\prod_{j=1}^{N} T_j)$ enumerations. This may be substantially reduced by a *dynamic programming* (DP) approach based on the recursion

$$V(S) = \max_{\substack{j \in S \\ 1 \le t \le T_j}} \left[R_j(t) + \beta^t V(S \setminus \{j\}) \right],\ S \in 2^J. \tag{3}$$

In (3), $V(S)$ is the total expected reward associated with an optimal fixed-time schedule for subset $S \subseteq J$. Please note that, if there are several t maximisers in (3) for some j then we shall *always* choose the largest. Breaking ties between distinct j, k is done arbitrarily. A DP approach certainly reduces the computational load but it remains exponential in N. We shall show in Section 5 that the search for an optimal fixed-time schedule may be regarded as a restless bandit problem, and consequently likely to be intractable in general. While we are able to develop a range of characterisations of optimal fixed-time schedules in later sections, heuristic approaches remain of interest and are also discussed.

We proceed to introduce ideas and results which will play an important role in subsequent discussions.

Lemma 1 *If $S_1 \supset S_2$ then $V(S_1) > V(S_2)$.*

Proof Denote by $\alpha(S_2)$ an optimal fixed-time schedule for S_2. Any fixed-time schedule for S_1 constructed as a concatenation of $\alpha(S_2)$ with a fixed-time schedule for $S_1 \setminus S_2$ with positive expected reward has total expected reward greater than $V(S_2)$. The result follows trivially. □

The following definition introduces notions of reward rate which will play a prominent role in the development.

Definition 1 *The **Gittins index** $G_j : \{0, 1, \ldots, T_j\} \to \mathbb{R}^+$ is given by*

$$G_j(s) = \max_{1 \le t \le T_j - s} \{G_j(s,t)\},\ 0 \le s \le T_j - 1, \tag{4}$$

where

$$G_j(s,t) = \beta^{-s}\{R_j(s+t) - R_j(s)\}(1-\beta^t)^{-1},\ 1 \le t \le T_j - s,\ 0 \le s \le T_j - 1.$$

*We set $G_j(T_j) = 0$ and call $G_j(0)$ the **initial index**.*

*The set of **maximal times** for job j is denoted $Max(j)$ and is a subset of $\{1, 2, \ldots, T_j\}$ defined by*

$$Max(j) = \left\{t;\ G_j(0,t) > G_j(0,s),\ t < s \le T_j\right\}.$$

*The set of **minimal times** for job j is denoted $Min(j)$ and is a subset of $\{1, 2, \ldots, T_j\}$ defined by*

$$Min(j) = \left\{t;\ G_j(t) < G_j(s),\ 0 \le s \le t-1\right\}$$

We write $\hat{t}_j$ for the largest t-value achieving the maximum in (4) when $s = 0$, i.e.

$$\hat{t}_j = \max\{t;\ G_j(0,t) = G_j(0)\}.$$

Proposition 2 *For all j we have*

$$\{\hat{t}_j, T_j\} \subseteq Min(j) \subseteq Max(j).$$

Further $t \in Max(j) \Rightarrow \hat{t}_j \le t \le T_j$.

Proof That we have $T_j \in Min(j) \cap Max(j)$ and $\hat{t}_j \in Max(j)$ (and indeed $\hat{t}_j$ the smallest member of $Max(j)$) are trivial consequences of the above definitions. It is a consequence of Gittins index theory (see, for example, Gittins (1989)) that $\hat{t}_j$ may be characterised as

$$\hat{t}_j = \min\{t;\ G_j(t) < G_j(0)\}. \tag{5}$$

That $\hat{t}_j \in Min(j)$ is immediate from (5).

Suppose now that $t \in Min(j) \setminus \{\hat{t}_j, T_j\}$, assumed non-empty. Write $\sigma(t)$ for the number of minimal times for job j which are no larger than t. In general, we list these $\sigma(t)$ times as

$$s_1 < s_2 < \ldots < s_{\sigma(t)} = t$$

We write $s_0 = 0$. By definition of the quantities concerned we have

$$G_j(0,t)(1-\beta^t) = \sum_{r=0}^{\sigma(t)-1} G_j(s_r, s_{r+1}-s_r)(\beta^{s_r}-\beta^{s_{r+1}})$$

$$= \sum_{r=0}^{\sigma(t)-1} G_j(s_r)(\beta^{s_r}-\beta^{s_{r+1}}) \quad (6)$$

$$> G_j(t)(1-\beta^t). \quad (7)$$

Equation (6) follows from the fact that, by slight extension of the result at (5), time $s_{r+1} - s_r$ achieves the index $G_j(s_r)$, $0 \leq r \leq \sigma(t) - 1$. Inequality (7) follows from the minimality of t. Hence we have from (7) and by definition of the Gittins index that

$$G_j(0,t) > G_j(t) \geq G_j(t,u), \ 1 \leq u \leq T_j - t. \quad (8)$$

It now follows simply from (8) that

$$G_j(0,t) > G_j(0,t+u), \ 1 \leq u \leq T_j - t,$$

and hence that $t \in Max(j)$. We conclude that $Min(j) \subseteq Max(j)$. This concludes the proof. □

The importance of the above ideas is highlighted by the following result.

Theorem 3 *Optimal fixed-time schedules contain only minimal times.*

Proof It is enough to show that any $\bar{t}$ achieving the maximum on the r.h.s. of (3) (i.e. the largest such maximiser for some accompanying $j \in S$) must be an element of $Min(j)$. We suppose that this is not the case and obtain a contradiction.

First note from Lemma 1 that, for the j, S concerned,

$$V(S) = R_j(\bar{t}) + \beta^{\bar{t}} V(S \setminus \{j\}) = G_j(0,\bar{t})(1-\beta^{\bar{t}}) + \beta^{\bar{t}} V(S \setminus \{j\}) > V(S \setminus \{j\})$$

from which it follows simply that

$$G_j(0,\bar{t}) > V(S \setminus \{j\}). \quad (9)$$

It must then follow that

$$G_j(\bar{t}) < V(S \setminus \{j\}) \quad (10)$$

since it is easy to establish that, if not, then the time r, $1 \leq r \leq T_j - \bar{t}$, achieving $G_j(\bar{t})$ satisfies

$$R_j(\bar{t}+r) + \beta^{\bar{t}+r} V(S \setminus \{j\}) = G_j(0,\bar{t}+r)(1-\beta^{\bar{t}+r}) + \beta^{\bar{t}+r} V(S \setminus \{j\})$$

$$\geq G_j(0,\bar{t})(1-\beta^{\bar{t}}) + \beta^{\bar{t}} V(S \setminus \{j\})$$

$$= R_j(\bar{t}) + \beta^{\bar{t}} V(S \setminus \{j\}). \quad (11)$$

Inequality (11) would contradict the status of $\bar{t}$ as the largest maximiser in (3).

If $\bar{t} \notin Min(j)$ then, there must exist u, $0 \leq u \leq \bar{t} - 1$ such that

$$G_j(u, \bar{t} - u) \leq G_j(u) \leq G_j(\bar{t}). \tag{12}$$

Since

$$G_j(0, u)(1 - \beta^u) + G_j(u, \bar{t} - u)(\beta^u - \beta^{\bar{t}}) = G_j(0, \bar{t})(1 - \beta^{\bar{t}}),$$

we have from (10) and (12) that

$$\begin{aligned} G_j(0, u)(1 - \beta^u) &\geq G_j(0, \bar{t})(1 - \beta^{\bar{t}}) - G_j(\bar{t})(\beta^u - \beta^{\bar{t}}) \\ &> G_j(0, \bar{t})(1 - \beta^{\bar{t}}) - V(S \setminus \{j\})(\beta^u - \beta^{\bar{t}}), \end{aligned}$$

from which we conclude that

$$\begin{aligned} R_j(u) + \beta^u V(S \setminus \{j\}) &= G_j(0, u)(1 - \beta^u) + \beta^u V(S \setminus \{j\}) \\ &> G_j(0, \bar{t})(1 - \beta^{\bar{t}}) + \beta^{\bar{t}} V(S \setminus \{j\}) \\ &= R_j(\bar{t}) + \beta^{\bar{t}} V(S \setminus \{j\}). \end{aligned} \tag{13}$$

Inequality (13) contradicts the status of $\bar{t}$ as a maximiser in (3). This concludes the proof. □

Comments

1. The problem of processing the jobs in J, giving each one a single uninterrupted interval of service, to maximise the total expected reward may be formulated as a discounted *Markov Decision Process* with finite state and action spaces. In such a formulation, the fixed-time schedules are precisely the stationary Markov policies. By standard theory (see, for example, Puterman (1994)) the search for an optimal policy may be restricted to the fixed-time schedules.
2. The modification of the above problem which allows the service of each job to be interrupted and to be resumed (from the same point) later may be formulated as a *multi-armed bandit problem* for which Gittins index policies are optimal. See, for example, Gittins (1989).
3. The results of the paper apply without further condition (beyond (a) above) on the distribution of the processing requirements of the jobs. Extensions of much of the material to cases where processing times are discrete with infinite support and to where they are absolutely continuous are available. Such extensions are bought, however, at the price of greater mathematical complexity in the results and analyses. We have chosen not to present such material here to enable the reader to focus on the core ideas.
4. The interest in maximal times is both theoretical and practical. We infer from Proposition 2 and Theorem 3 that it must be true that optimal fixed-time schedules contain only maximal times. This information is used in the subsequent theory. Also, maximal times are easy to obtain and it is often the case that $Max(j) \setminus Min(j)$ is small or even empty. It is possible to elucidate job structures where the latter is the case.

5. The model in (a)–(c) may be elaborated to provide for the payment of a (discounted) *set-up cost* when the processing of job j begins and a (discounted) *tear-down cost* payable when its processing is terminated. Under the condition that it remains possible to make a profit (in expectation) from each job, almost all of the paper's results are capable of generalisation to accommodate such costs.

3 Batches of identical jobs

Considerable simplifications result in the important special case in which all jobs in the batch J are identical (i.e., have the same r_j and $p_j(\cdot)$). We may thus drop the job identifier j from the notation. The importance of the initial Gittins index leads use to abbreviate the notation $G(0)$ to G with $\hat{t}$ the largest t-value achieving it. Plainly in such cases the ordering of the jobs is immaterial and a fixed-time schedule is simply a vector $\mathbf{t}$ of processing times. The DP recursion in (3) now becomes

$$V(n) = \max_{1 \le t \le T} \{R(t) + \beta^t V(n-1)\}, \ 1 \le n \le N, \tag{14}$$

where $V(n)$ is the maximal return available from a batch of n jobs. We write $t(n)$ for the largest maximiser in (14). Hence $t(n)$ has the interpretation as the allocated processing time (in an optimal fixed-time schedule) when n jobs remain to be processed.

Theorem 4 characterises the two sequences, $\{V(n), n \in \mathbb{Z}^+\}$ and $\{t(n), n \in \mathbb{Z}^+\}$ which yield the solutions to (14) for all $N \in \mathbb{Z}^+$.

Theorem 4 *(a)* $\{V(n), n \in \mathbb{Z}^+\}$ *is a strictly increasing sequence with* $V(1) = R(T)$ *and*

$$\lim_{n \to \infty} V(n) = G;$$

(b) $\{t(n), n \in \mathbb{Z}^+\}$ *is a decreasing sequence of minimal values with* $t(1) = T$ *and*

$$\lim_{n \to \infty} t(n) = \hat{t}.$$

Proof We first prove (a). The strictly increasing nature of the sequence $\{V(n), n \in \mathbb{Z}^+\}$ is an immediate consequence of Lemma 1. It is trivial that $V(1) = R(T)$. Now observe that

$$\begin{aligned} V(n) = R\{0, t(n)\} + \beta^{t(n)} V(n-1) &= G\{0, t(n)\}\{1 - \beta^{t(n)}\} + \beta^{t(n)} V(n-1) \\ &< G\{1 - \beta^{t(n)}\} + \beta^{t(n)} V(n), \end{aligned}$$

from which it follows that

$$V(n) < G, \ n \in \mathbb{Z}^+. \tag{15}$$

From (15) and the increasing nature of $\{V(n), n \in \mathbb{Z}^+\}$, the limit in (a) must exist with

$$\lim_{n \to \infty} V(n) \le G. \tag{16}$$

Now consider the fixed-time schedule which gives $\hat{t}$ units of processing to each of n jobs. The expected reward from this schedule is

$$R(\hat{t})\sum_{j=0}^{n-1}\beta^{j\hat{t}} = G(0,\hat{t})(1-\beta^{\hat{t}})\sum_{j=0}^{n-1}\beta^{j\hat{t}} = G(1-\beta^{n\hat{t}})$$

and hence

$$V(n) \geq G(1-\beta^{n\hat{t}}),\ n \in \mathbb{Z}^+,$$

from which it follows that

$$\lim_{n\to\infty} V(n) \geq G. \tag{17}$$

Theorem 4(a) now follows from (16) and (17).

We now proceed to (b). Suppose that $\{t(n), n \in \mathbb{Z}^+\}$ is not decreasing and obtain a contradiction. Hence there exists m for which $t(m+1) > t(m)$. Since $t(m)$ is minimal (Theorem 3), and hence maximal (Proposition 2) it must follow that

$$G\{0, t(m)\} \geq G\{0, t(m+1)\}$$

and hence that

$$\begin{aligned} R\{t(m)\} + \beta^{t(m)}R\{t(m+1)\} &= G\{0,t(m)\}\{1-\beta^{t(m)}\} \\ &\quad +\beta^{t(m)}G\{0,t(m+1)\}\{1-\beta^{t(m+1)}\} \\ &\geq G\{0,t(m+1)\}\{1-\beta^{t(m+1)}\} \\ &\quad +\beta^{t(m+1)}G\{0,t(m)\}\{1-\beta^{t(m)}\} \\ &= R\{t(m+1)\} + \beta^{t(m+1)}R\{t(m)\}. \end{aligned} \tag{18}$$

If the inequality in (18) is strict then, it follows simply that in a problem with $m+1$ jobs the ordered set of allocated processing times $\{t(m), t(m+1), t(m-1), \ldots, t(1)\}$ has a larger expected reward than does $\{t(m+1), t(m), t(m-1), \ldots, t(1)\}$. This contradicts the supposed optimality of the latter. If we have equality through (18) then these two ordered sets of times have the same expected reward. But the inferred optimality of $\{t(m), t(m+1), \ldots, t(1)\}$ together with $t(m+1) > t(m)$ contradicts the choice of $t(m)$ as the largest maximiser in (14) when $n = m$. Hence $\{t(n), n \in \mathbb{Z}^+\}$ must be decreasing.

From Proposition 2 and Theorem 3 we have that

$$t(n) \in Max \Rightarrow t(n) \geq \hat{t},\ n \in \mathbb{Z}^+. \tag{19}$$

Theorem 3 further asserts that all members of the sequence are members of Min. It is trivial that $t(1) = T$. All that now remains for part (b) is the identification of the limit.

We suppose now that

$$\lim_{n\to\infty} t(n) = \bar{t} > \hat{t} \tag{20}$$

and obtain a contradiction. From (20) it would follow that $t(n) = \bar{t},\ n \geq \bar{N}$ for some $\bar{N} \in \mathbb{Z}^+$ and hence that

$$V(\bar{N}+m) = R(\bar{t}) \sum_{j=0}^{m-1} \beta^{j\bar{t}} + \beta^{m\hat{t}} V(\bar{N})$$
$$= G(0,\bar{t})(1-\beta^{m\bar{t}}) + \beta^{m\bar{t}} V(\bar{N}),\ m \in \mathbb{N}.$$

From this we infer that

$$\lim_{n\to\infty} V(n) = \lim_{m\to\infty} V(\bar{N}+m) = G(0,\bar{t}) < G(0,\hat{t}) = G$$

which contradicts Theorem 4(a). This concludes the proof. □

Since from Theorem 4 the sequence $\{t(n), n \in \mathbb{Z}^+\}$ is decreasing and consists only of minimal values, we need only determine the smallest and largest batch sizes (n) for which each member of Min equals $t(n)$. Proposition 5 assists this determination by bounding the range of n for which each minimal time can be optimal.

To assist the development, list the minimal times in ascending order as

$$\hat{t} = t_{(1)} < t_{(2)} < \ldots < t_{(M)} = T$$

where $M = |Min|$. We write

$$\Gamma_r = \{n;\ n \in \mathbb{Z}^+ \text{ and } t(n) = t_{(r)}\},\ 1 \leq r \leq M,$$

for the optimality ranges of interest. Finally, we use Θ_r for the smallest integer greater than

$$\frac{\ln\left([G\{0,t_{(r)}\} - G\{t_{(r)}\}]/G\{0,t_{(r)}\}\right)}{t_{(r)} \ln \beta} + 1,\ 1 \leq r \leq M. \tag{21}$$

Note that in (21) both logarithms are negative and hence $\Theta_r \geq 2,\ 1 \leq r \leq M$.

Proposition 5 *(a)* $\Gamma_1 \supseteq [\Theta_1, \infty)$;
(b) $\Gamma_r \subseteq [1, (\min_{1\leq s\leq r-1} \Theta_s) - 1],\ 2 \leq r \leq M$.

Proof Fix $1 \leq r \leq M-1$. By the definition of Θ_r we have that

$$G\{0,t_{(r)}\}\{1-\beta^{(n-1)t_{(r)}}\} > G\{t_{(r)}\},\ n \geq \Theta_r,$$

and hence from Definition 1 that

$$G\{0,t_{(r)}\}\{1-\beta^{(n-1)t_{(r)}}\}\{\beta^{t_{(r)}} - \beta^t\} > R(0,t) - R\{0,t_{(r)}\},$$
$$T \geq t \geq t_{(r)} + 1,\ n \geq \Theta_r. \tag{22}$$

However, since $G\{0,t_{(r)}\}\{1-\beta^{(n-1)t_{(r)}}\}$ is the expected reward when each of $n-1$ jobs is processed for $t_{(r)}$ time units then

$$V(n-1) \geq G\{0,t_{(r)}\}\{1-\beta^{(n-1)t_{(r)}}\}$$

and hence from (22) that

$$R\{0, t_{(r)}\} + \beta^{t_{(r)}} V(n-1) > R(0,t)\beta^t + \beta^t V(n-1),$$
$$T \geq t \geq t_{(r)} + 1,\ n \geq \Theta_r. \tag{23}$$

By taking $r = 1$ in (23) then appeal to Theorem 4(b) guarantees that $t(n) = t_{(1)} = \hat{t}$ for the range $n \geq \Theta_1$. Part (a) follows. For $2 \leq r \leq M$ we conclude that we *cannot* have $t(n) = t_{(r)}$ when $n \geq \Theta_s$ for some $s \leq r - 1$. Part (b) is an immediate consequence. This concludes the proof. □

Comment

To illustrate this and succeeding material we introduce four different job types, each having unit reward and $T = 10$. The discount rate β is assumed to be $e^{-0.05}$ throughout. For job types 1 and 2 the probabilities $p(s)$, $1 \leq s \leq 10$, were obtained by sampling independently from a uniform $U(2, 5)$ distribution and normalising. For job types 3 and 4 the probabilities $p(s)$, $1 \leq s \leq 10$, were obtained by sampling independently from an exponential distribution with mean 0.5 and normalising. The details (given correct to 3 d.p.) are recorded in Table 1(a)–(d).

Hence, for example, from Table 1(d) for job type 4 the minimal times are 7, 8 and 10. This is also the set of maximal times. There are no n for which $t(n) = 8$ and so its optimality range is empty. We have $t(n) = 10$, $n = 1,\ 2$ and $t(n) = 7$, $n \geq 3$. It follows that the optimal fixed-time schedule for a batch of N such jobs where $N \geq 2$ will give each of the first $N-2$ jobs to be scheduled seven units of processing while the final two jobs will each receive ten units.

Finally, to assist the reader in gauging the degree of conservatism in Proposition 5, observe that the Θ_1 values for our four job types are 27, 29, 7 and 3 for types 1, 2, 3 and 4 respectively. Hence Proposition 5(a) asserts that the optimal range for $\hat{t}$ in the four cases contains $[27, \infty)$, $[29, \infty)$, $[7, \infty)$ and $[3, \infty)$. Thus from Table 1(d) the exact optimality range is identical to the bound given in Proposition 5(a) for job type 4 while from Table 1(a) for job type 1 the proposition is conservative and the exact range is $[8, \infty)$.

4 Distinct job classes

In Section 3 all jobs were assumed identical and so attention was focused exclusively on the vector of allocated processing times. We develop the theory by now considering batches of jobs in which each job belongs to one of C distinct classes. We reinstate the suffix j, though it will now act as a class identifier rather than a job identifier as in Section 2. Hence all jobs in class j share the same reward r_j and service time distribution $p_j(\cdot)$. We also have the initial and general class indices $G_j = G_j(0)$ and $G_j(s)$ respectively by obvious extension from Definition 1. We shall suppose that classes have distinct initial Gittins indices and number them such that

$$G_1 > G_2 > \ldots > G_C > 0. \tag{24}$$

Determination of an optimal fixed-time schedule now has to consider the order in which jobs are processed in addition to the processing times which they are

Table 1 Details for four job types, including indices, maximal and minimal times and optimality ranges

s	1	2	3	4	5	6	7	8	9	10
$p(s)$	0.132	0.094	0.076	0.101	0.110	0.088	0.068	0.108	0.127	0.097
$G(0, s)$	2.582	2.213	1.983	1.979	2.007	1.965	1.886	1.908	1.960	1.955
$G(s-1)$	2.852	1.866	1.872	2.047	2.136	1.887	1.948	2.286	2.843	1.892

Maximal times $\{1, 2, 5, 6, 9, 10\}$

Minimal times	1	10
Optimality ranges	$[8, \infty)$	$[1, 7]$

Table 1(a): Details for jobs of type 1.

s	1	2	3	4	5	6	7	8	9	10
$p(s)$	0.106	0.089	0.132	0.078	0.119	0.127	0.080	0.095	0.111	0.062
$G(0, s)$	2.075	1.911	2.123	1.984	2.045	2.107	2.042	2.022	2.034	1.969
$G(s-1)$	2.123	2.149	2.580	2.089	2.395	2.469	1.851	2.002	2.158	1.214

Maximal times $\{3, 6, 7, 9, 10\}$

Minimal times	3	6	9	10
Optimality ranges	$[16, \infty)$	$[7, 15]$	$[4, 6]$	$[1, 3]$

Table 1(b): Details for jobs of type 2.

s	1	2	3	4	5	6	7	8	9	10
$p(s)$	0.097	0.072	0.247	0.032	0.125	0.080	0.013	0.029	0.019	0.287
$G(0, s)$	1.882	1.647	2.653	2.183	2.229	2.130	1.902	1.763	1.636	1.949
$G(s-1)$	2.653	3.068	4.821	1.563	2.441	1.589	1.599	2.091	2.192	5.590

Maximal times $\{3, 5, 6, 10\}$

Minimal times	3	10
Optimality ranges	$[5, \infty)$	$[1, 4]$

Table 1(c): Details for jobs of type 3.

s	1	2	3	4	5	6	7	8	9	10
$p(s)$	0.011	0.127	0.189	0.101	0.137	0.050	0.210	0.058	0.010	0.107
$G(0, s)$	0.206	1.313	2.066	2.045	2.159	1.985	2.243	2.126	1.952	1.963
$G(s-1)$	2.243	3.068	3.690	2.400	2.679	2.491	4.088	1.128	1.122	2.091

Maximal times $\{7, 8, 10\}$

Minimal times	7	8	10
Optimality ranges	$[3, \infty)$	$\emptyset$	$[1, 2]$

Table 1(d): Details for jobs of type 4.

allocated. It may be conjectured from (24) that any optimal fixed-time schedule will process all class 1 jobs first, followed by class 2 and so on. While this is not the case in general (and we shall present a counterexample later in the section) we shall present conditions which are sufficient for the assertion to be true.

Before we proceed to the theory, we develop some notation. Vector $\mathbf{n} = (n_1, n_2, \ldots, n_C)$ will represent a problem (state) in which n_j jobs of class j

are present, $1 \leq j \leq C$. In what follows $\mathbf{n}$ may either present a *problem* (jobs to be processed at 0) or a state (jobs currently remaining to be processed) in some larger problem. If we wish to focus attention on one particular class, k say, then we may re-write $\mathbf{n}$ as $(\mathbf{n}_{k-1}, n_k, \mathbf{n}^{k+1})$ where $\mathbf{n}_{k-1}$ and $\mathbf{n}^{k+1}$ represent the sub-vectors $(n_1, n_2, \ldots, n_{k-1})$ and $(n_{k+1}, \ldots, n_C)$ respectively. Sub-vector $\mathbf{n}_{k-1}$ is empty when $k = 1$ as is $\mathbf{n}^{k+1}$ when $k = C$. We shall use $\mathbf{e}_j$ to denote a vector whose j^{th} component is 1 with zeroes elsewhere and $\mathbf{0}$ for a zero vector. We combine these notations in ways whose meanings should be clear from the context. For example, if $1 \leq j \leq k-1$ then we use $((\mathbf{e}_j)_{k-1}, n_k, \mathbf{n}^{k+1})$ to denote a situation in which a single class j job is the only one present from classes 1 to $k-1$. Any allocated processing time for a class j job is called a *j-time*.

In the current multi-class context it will be most convenient to represent a general fixed-time schedule by a pair $(\nu, \mathbf{t})$, with ν an ordered set of job classes and $\mathbf{t}$ a vector of allocated processing times. Under $(\nu, \mathbf{t})$ a job from class ν_1 is processed first (at time 0) for time t_1, followed by a job from class ν_2 for time t_2, and so on. We write $V_{\nu,\mathbf{t}}(\mathbf{n})$ for the total expected reward which results when $(\nu, \mathbf{t})$ is used for problem $\mathbf{n}$ and $V(\mathbf{n})$ for the largest such reward. We denote any fixed-time schedule achieving the latter $(\nu^*, \mathbf{t}^*)$, under the usual assumptions (i.e. largest maximising times always chosen).

Proposition 6 is an extension of part of Theorem 4(b) and is appropriate for the multi-class context. The proof is similar and is omitted.

Proposition 6 *In any optimal fixed-time schedule* $(\nu^*, \mathbf{t}^*)$

$$\nu_k^* = \nu_l^*, \; k < l \Rightarrow t_k^* \leq t_l^*.$$

The assertions of Proposition 7 are straightforward consequences of Lemma 1 and Theorem 3.

Proposition 7 *(a)* $V(\mathbf{n})$ *is strictly increasing componentwise;*
(b) For any problem $\mathbf{n}$, $t_k^* \in Min(\nu_k^*)$, $1 \leq k \leq \sum_{j=1}^{C} n_j$.

The characterisations of value functions which follow in Proposition 8 have major implications for what we can say about how jobs (classes) are ordered in optimal fixed-time schedules.

Proposition 8 *(a) If* $\sum_{j=1}^{k-1} n_j \geq 1$ *then*

$$\lim_{n_k \to \infty} V(\mathbf{n}_{k-1}, n_k, \mathbf{n}^{k+1}) > G_k, \; \forall \mathbf{n}^{k+1}, \; 2 \leq k \leq C;$$

(b)

$$\lim_{n_k \to \infty} V(\mathbf{0}_{k-1}, n_k, \mathbf{n}^{k+1}) = G_k, \; \forall \mathbf{n}^{k+1}, \; 1 \leq k \leq C.$$

Proof By Proposition 7(a), $\{V(\mathbf{n}_{k+1}, n_k, \mathbf{n}^{k+1}), n_k \in \mathbb{N}\}$ is a strictly increasing sequence, for any choice of $\mathbf{n}_{k-1}$, $\mathbf{n}^{k+1}$. Further, the expected reward earned by general fixed-time schedule $(\nu, \mathbf{t})$ is given by

$$V_{\nu,\mathbf{t}}(\mathbf{n}_{k-1}, n_k, \mathbf{n}^{k+1}) \equiv \sum_{l=1}^{N} G_{\nu_l}(0, t_l)(1-\beta^{t_l})\beta^{\sum_{m=1}^{l-1} t_m}$$
$$\leq \sum_{l=1}^{N} G_{\nu_l}(1-\beta^{t_l})\beta^{\sum_{m=1}^{l-1} t_m}$$
$$\leq G_1(1-\beta^{\sum_{l=1}^{N} t_l}) < G_1, \tag{25}$$

where $N = \sum_{j=1}^{C} n_j$ is the total number of jobs to be processed. The inequalities in (25) make use of Definition 1 and (24). It follows that all sequences $\{V(\mathbf{n}_{k-1}, n_k, \mathbf{n}^{k+1}), n_k \in \mathbb{N}\}$ are bounded above by G_1. Hence the limits in the statement of the proposition exist.

For part (a), for problem $\mathbf{n}$ with $\sum_{j=1}^{k-1} n_j \geq 1$ for some k, $2 \leq k \leq C$, consider the fixed-time schedule $(\tilde{\nu}, \tilde{\mathbf{t}})$ which first processes all class 1 jobs and which gives each one $\hat{t}_1$ units of processing, then processes all class 2 jobs for time $\hat{t}_2$, and so on. The corresponding total expected reward is given by

$$V_{\tilde{\nu},\tilde{\mathbf{t}}}(\mathbf{n}_{k-1}, n_k, \mathbf{n}^{k+1}) = \sum_{i=1}^{C} G_i(0, \hat{t}_i)(1-\beta^{n_i\hat{t}_i})\beta^{\sum_{j=1}^{i-1} n_j\hat{t}_j}$$
$$\geq \sum_{i=1}^{k-1} G_i(1-\beta^{n_i\hat{t}_i})\beta^{\sum_{i=1}^{j-1} n_i\hat{t}_i}$$
$$+G_k(1-\beta^{n_k\hat{t}_k})\beta^{\sum_{i=1}^{k-1} n_i\hat{t}_i} \tag{26}$$

From (25) and (26) we conclude that if $\sum_{j=1}^{k-1} n_j \geq 1$ then

$$G_k < \lim_{n_k\to\infty} V_{\tilde{\nu},\tilde{\mathbf{t}}}(\mathbf{n}_{k-1}, n_k, \mathbf{n}^{k+1}) \leq \lim_{n_k\to\infty} V(\mathbf{n}_{k-1}, n_k, \mathbf{n}^{k+1})$$

as required. This proves part (a). The proof of part (b) is similar and is omitted. □

We are able to deploy Proposition 8 to make statements about how the processing of jobs is ordered in a fixed-time schedule.

Theorem 9 *For each class $k \geq 2$ there exists $N_k \in \mathbb{Z}^+$ such that $n_k \geq N_k$ implies that for all choices of $\mathbf{n}_{k-1}$, $\mathbf{n}^{k+1}$ every optimal fixed-time schedule for $(\mathbf{n}_{k-1}, n_k, \mathbf{n}^{k+1})$ processes all jobs from classes 1 to $k-1$ before any jobs from classes k to C.*

Proof We suppose that $2 \leq k \leq C$ and define N_k as follows:

$$N_k = \min\left\{n_k; n_k \in \mathbb{Z}^+ \text{ and } V((\mathbf{e}_j)_{k-1}, n_k, \mathbf{0}^{k+1}) > G_k, 1 \leq j \leq k-1\right\}. \tag{27}$$

From Proposition 8(a), N_k is well defined and finite.

Now suppose that $n_k \geq N_k$ and consider some problem $(\mathbf{n}_{k-1}, n_k, \mathbf{n}^{k+1})$ for which $\sum_{j=1}^{k-1} n_j \geq 1$. We suppose that there exists an optimal fixed-time schedule $(\nu^*, \mathbf{t}^*)$ which processes a job from one of the classes k to C at time 0 and obtain a contradiction. By (27) and Proposition 7(a) we must have

$$V(\mathbf{n}_{k-1}, n_k, \mathbf{n}^{k+1}) > G_k. \tag{28}$$

Using the expression in (25) for the total expected reward earned from $(\nu^*, \mathbf{t}^*)$, we infer from (28) that

$$\begin{aligned} V(\mathbf{n}_{k-1}, n_k, \mathbf{n}^{k+1}) &= V_{\nu^*,\mathbf{t}^*}(\mathbf{n}_{k-1}, n_k, \mathbf{n}^{k+1}) \\ &= \sum_{l=1}^{N} G_{\nu_l^*}(0, t_l^*)(1-\beta^{t_l^*})\beta^{\sum_{m=1}^{l-1} t_m^*} > G_k. \end{aligned} \tag{29}$$

It follows from (29) that there must exist l, $1 \le l \le N$, such that

$$G_{\nu_l^*}(0, t_l^*) > G_k. \tag{30}$$

Denote by L the smallest such. However,

$$G_{\nu_L^*}(0, t_L^*) > G_k \Rightarrow G_{\nu_L^*} > G_k$$

from which we conclude that ν_L^* must be one of the classes 1 to $k-1$. Since $(\nu^*, \mathbf{t}^*)$ processes a job from one of the classes k to C at time 0, it must follow that $L \ge 2$.

We now construct a new fixed-time schedule which modifies $(\nu^*, \mathbf{t}^*)$ by a pairwise interchange which promotes the processing (ν_L^*, t_L^*) to time 0, but which otherwise keeps the processing schedule and times in $(\nu^*, \mathbf{t}^*)$ unchanged. The expected reward from this new schedule is given by

$$\begin{aligned} &G_{\nu_L^*}(0, t_L^*)(1-\beta^{t_L^*}) + \beta^{t_L^*}\left\{\sum_{l=1}^{L-1} G_{\nu_l^*}(0, t_l^*)(1-\beta^{t_l^*})\beta^{\sum_{m=1}^{l-1} t_m^*}\right\} \\ &+ \sum_{l=L+1}^{N} G_{\nu_l^*}(0, t_l^*)(1-\beta^{t_l^*})\beta^{\sum_{m=1}^{l-1} t_m^*}. \end{aligned} \tag{31}$$

However, by the characterisation of L it follows that

$$G_{\nu_l^*}(0, t_l^*) \le G_k, \quad 1 \le l \le L-1$$

and hence that

$$\sum_{l=1}^{L-1} G_{\nu_l^*}(0, t_l^*)(1-\beta^{t_l^*})\beta^{\sum_{m=1}^{l-1} t_m^*} \le G_k(1-\beta^{\sum_{m=1}^{L-1} t_m^*}), \tag{32}$$

while

$$G_{\nu_L^*}(0, t_L^*)(1-\beta^{t_L^*}) > G_k(1-\beta^{t_L^*}). \tag{33}$$

It is now a simple matter to infer from (32) and (33) that the expression in (31) is strictly greater than that in (29). This contradicts the optimality of the proposed $(\nu^*, \mathbf{t}^*)$.

We conclude that for any problem $(\mathbf{n}_{k-1}, n_k, \mathbf{n}^{k+1})$ with $n_k \ge N_k$ the first job to be processed in an optimal fixed-time schedule must come from classes 1 to $k-1$, provided there is one. Repetition of this argument yields the conclusion that all jobs from classes 1 to $k-1$ must be processed before any others. This proves the result. □

Corollary 10 *For each class $k \geq 2$ there exists $N_k \in \mathbb{Z}^+$ such that $n_k \geq N_k$, $2 \leq k \leq C$, implies that all optimal fixed-times schedules for problem* $\mathbf{n}$ *process jobs in decreasing order of their class identifier (i.e., all class 1 jobs are processed first, then class 2 jobs and so on).*

Proof If the N_k are as defined in (27) then it is the conclusion of Theorem 9 that, under the hypothesis of the corollary, any optimal fixed-time schedule must process 1 to $k-1$ before any others, for every k in the range $2 \leq k \leq C$. This can only happen if jobs are processed in decreasing order of their class identifier, as required. □

Consider now some problem $(\mathbf{n}_{k-1}, 0, \mathbf{0}^{k+1})$ populated by jobs from classes 1 to $k-1$ only, for some $k \geq 2$. Let $(\nu, \mathbf{t})$ be an arbitrarily chosen fixed-time schedule for this problem. We shall use the contracted notation $V_{\nu,\mathbf{t}}(\mathbf{n}_{k-1})$ to denote the total expected reward under $(\nu, \mathbf{t})$ and $T_{\nu,\mathbf{t}}(\mathbf{n}_{k-1})$ for the total time taken for its implementation. Corollary 11 is a development of the key Proposition 8.

Corollary 11 *(a) For all $k \geq 2$, $\mathbf{n}_{k-1}$, $\mathbf{n}^{k+1}$*

$$\lim_{n_k \to \infty} V(\mathbf{n}_{k-1}, n_k, \mathbf{n}^{k+1}) = \max_{\nu,\mathbf{t}} \left\{ V_{\nu,\mathbf{t}}(\mathbf{n}_{k-1}) + \beta^{T_{\nu,\mathbf{t}}(\mathbf{n}_{k-1})} G_k \right\} \equiv W(\mathbf{n}_{k-1}) \tag{34}$$

where the maximisation in (34) is over all fixed-time schedules for $(\mathbf{n}_{k-1}, 0, \mathbf{0}^{k+1})$;
(b) Function $W : (\mathbb{N})^{k-1} \to \mathbb{R}^+$ in (a) is increasing componentwise with $W(\mathbf{0}_{k-1}) = G_k$.

Proof We suppose that $\epsilon > 0$, $2 \leq k \leq C$ and define $\bar{N}_k(\epsilon)$ as follows:

$$\bar{N}_k(\epsilon) = \min \left\{ n_k;\, n_k \in \mathbb{Z}^+ \text{ and } V(\mathbf{0}_{k-1}, n_k, \mathbf{0}^{k+1}) > G_k - \epsilon \right\}. \tag{35}$$

From Proposition 8(b), $\bar{N}_k(\epsilon)$ is well defined and finite. We now introduce

$$N_k^*(\epsilon) = \max\{N_k, \bar{N}_k(\epsilon)\},$$

with N_k given in (27). Invoking Theorem 9 we conclude that for any choice of $\mathbf{n}_{k-1}$, $\mathbf{n}^{k+1}$,

$$\begin{aligned} n_k \geq N_k^*(\epsilon) \Rightarrow V(\mathbf{n}_{k-1}, n_k, \mathbf{n}^{k+1}) &\geq V_{\nu,\mathbf{t}}(\mathbf{n}_{k-1}) \\ &\quad + \beta^{T_{\nu,\mathbf{t}}(\mathbf{n}_{k-1})} V(\mathbf{0}_{k-1}, n_k, \mathbf{n}^{k+1}) \\ &\geq V_{\nu,\mathbf{t}}(\mathbf{n}_{k-1}) + \beta^{T_{\nu,\mathbf{t}}(\mathbf{n}_{k-1})}(G_k - \epsilon) \end{aligned} \tag{36}$$

for any fixed-time schedule $(\nu, \mathbf{t})$ for $(\mathbf{n}_{k-1}, 0, \mathbf{0}^{k+1})$. Note that the second inequality in (36) follows from (35) and Proposition 7(a). It then follows that

$$\lim_{n_k \to \infty} V(\mathbf{n}_{k-1}, n_k, \mathbf{n}^{k+1}) \geq \max_{\nu,\mathbf{t}} \left\{ V_{\nu,\mathbf{t}}(\mathbf{n}_{k-1}) + \beta^{T_{\nu,\mathbf{t}}(\mathbf{n}_{k-1})} G_k \right\}. \tag{37}$$

However, from Theorem 9 it follows that for each $\mathbf{n}_{k-1}$, $\mathbf{n}^{k+1}$ and $n_k \geq N_k$ there exists some $(\nu, \mathbf{t})$ for which

$$\begin{aligned} V(\mathbf{n}_{k-1}, n_k, \mathbf{n}^{k+1}) &= V_{\nu,\mathbf{t}}(\mathbf{n}_{k-1}) + \beta^{T_{\nu,\mathbf{t}}(\mathbf{n}_{k-1})} V(\mathbf{0}_{k-1}, n_k, \mathbf{n}^{k+1}) \\ &\leq V_{\nu,\mathbf{t}}(\mathbf{n}_{k-1}) + \beta^{T_{\nu,\mathbf{t}}(\mathbf{n}_{k-1})} G_k, \end{aligned} \tag{38}$$

from Propositions 7(a) and 8(b). Part (a) of the result now follows from (37) and (38). Part (b) is a simple consequence of Proposition 8(b). This concludes the proof. □

We conclude this theoretical discussion by proposing an elaboration of Theorem 9 which gives important further information concerning the way in which jobs are sequenced in optimal fixed-time schedules.

Proposition 12 *For each* $k \geq 1$ *there exists* $\tilde{N}_k \in \mathbb{Z}^+$ *such that* $n_k \geq \tilde{N}_k$ *implies that for all choices of* $\mathbf{n}_{k-1}$, $\mathbf{n}^{k+1}$ *every optimal fixed-time schedule for* $(\mathbf{n}_{k-1}, n_k, \mathbf{n}^{k+1})$ *must* first *process all jobs from classes 1 to* $k-1$ *and will then process at least* $n_k - \tilde{N}_k + 1$ *jobs from class k before proceeding further.*

Proof For any $k \geq 1$ we define $\bar{N}_k$ as follows:

$$\bar{N}_k = \min\left\{n_k; n_k \in \mathbb{Z}^+ \text{ and } V(\mathbf{0}_{k-1}, n_k, \mathbf{0}^{k+1}) > G_{k+1}\right\}, \tag{39}$$

where in (39) we take $G_{C+1} = 0$ and hence $\bar{N}_C = 1$. From Proposition 8(b) and (24), $\bar{N}_k$ is well defined and finite. We now introduce

$$\tilde{N}_k = \begin{cases} \bar{N}_k, & k = 1, \\ \max\{N_k, \bar{N}_k\}, & 2 \leq k \leq C, \end{cases} \tag{40}$$

where N_k is as in (27).

Consider now any state of the form $(\mathbf{0}_{k-1}, n_k, \mathbf{n}^{k+1})$. From Proposition 7(a) and (39) we conclude that, for any $n_k \geq \tilde{N}_k$

$$V(\mathbf{0}_{k-1}, n_k, \mathbf{n}^{k+1}) > G_{k+1} \quad \forall \mathbf{n}^{k+1}. \tag{41}$$

An argument similar to that in the proof of Theorem 9 from (28) yields the conclusion that if $n_k \geq \tilde{N}_k$ then in any problem $(\mathbf{0}_{k-1}, n_k, \mathbf{n}^{k+1})$ the first job to be processed must come from class k. The result now follows by an appeal to Theorem 9. □

Comment

In order to crystallise further what has been established concerning the structure of optimal fixed-time schedules, we now specialise to the case $C = 2$. In our following example, we shall suppose that jobs in class 1 have the characteristics of job type 1 in Section 3 for which $G_1 = 2.582$, $\hat{t}_1 = 1$ and the minimal values are $\{1, 10\}$. Class 2 has the characteristics of job type 2 with $G_2 = 2.123$, $\hat{t}_2 = 3$ with minimal values $\{3, 6, 9, 10\}$. Note that $G_1 > G_2 > 0$ as is required for (24). We consider all such two class problems $(n_1, n_2) \in (\mathbb{Z}^+)^2$.

To establish the structure of optimal fixed-time schedules, key quantities are N_2 from (27) and $\bar{N}_1$ from (39). Direct calculation gives $\bar{N}_1 = 16$, $N_2 = 17$ in this case. Proposition 12 implies that for any state with $n_1 \geq 16$ it is optimal to process a class 1 job irrespective of the value of n_2. Theorem 9 further asserts that for any state with $n_2 \geq 17$ it is optimal to process a class 1 job irrespective of the value of n_1. Hence the only unresolved cases are those (n_1, n_2) for which $1 \leq n_1 \leq 15$

Table 2 The decision (class, allocated processing) taken by the optimal fixed-time schedule for state (n_1, n_2) in some two class problems

n_2 \ n_1	1	2	3	4	5	6	{7 – 15}
1	(2,10)	(2,9)	(2,9)	(2,9)	(2,9)	(2,6)	(1,1) optimal in all cases
2	(2,9)	(2,9)	(2,9)	(2,9)	(2,6)	(1,1)	
3	(2,9)	(2,9)	(2,9)	(2,6)	(2,6)	(1,1)	
4	(2,9)	(2,9)	(2,6)	(2,6)	(1,1)	(1,1)	
5	(2,9)	(2,6)	(2,6)	(1,1)	(1,1)	(1,1)	
6	(2,6)	(2,6)	(1,1)	(1,1)	(1,1)	(1,1)	
7	(2,6)	(1,1)	(1,1)	(1,1)	(1,1)	(1,1)	
8	(2,6)	(1,1)	(1,1)	(1,1)	(1,1)	(1,1)	
{9 – 16}	(1,1) optimal in all cases						(1,1) optimal in all cases

and $1 \leq n_2 \leq 16$. In Table 2 find values of the optimal decision (class, allocated processing) for these cases.

From Table 2, first note that all details are consistent with our theoretical results, including Propositions 6 and 7. Note that all allocated processing times are from the minimal set for each class and that these times are monotonic in the sense of Proposition 6. Note also the degree of conservatism in Theorem 9 and Proposition 12. From the table we see that the thresholds for the optimality of class 1 processing actually occur at $n_1 = 7$ and $n_2 = 9$, somewhat below the respective values $\tilde{N}_1 = 16$ and $N_2 = 17$ which emerge from the theoretical work. Note finally that it is evidently not true for this example that the processing of class 1 jobs is preferred to the processing of class 2 jobs in all states where both are available. Indeed it follows from Table 2 that in any problem (n_1, n_2) with $1 \leq n_2 \leq 8$ it will be true that the *last* job to be processed by the optimal fixed-time schedule will be from class 1 with allocated processing equal to ten. This is the counterexample to the false conjecture cited at the start of the section.

Note finally that, from (27) and (40) exact evaluation of the key bounding thresholds N_k, $\tilde{N}_k$ requires access to the value function V. Deployment of easily computed lower bounds on the value function of the form given in (26) can yield upper bounds on N_k, $\tilde{N}_k$ which are sometimes close to tight. This is in the spirit of Proposition 5.

5 Heuristic development

It is a feature of the fixed-time scheduling problems outlined in Section 2 that, once a job's allocated processing has been fully given it becomes ineligible for further

service effort. One way of viewing this is to observe that a job's status *may* change while it is *not* being processed. If a job is scheduled for processing at some time t but *not* at time $t+1$ then at time $t+2$ it *cannot* be a candidate for further service. It is this feature which means that the problem of constructing a fixed-time scheduling problem falls within the class of *restless bandit problems*, a family of intractable decision processes introduced by Whittle (1988). Restless bandit problems extend the class of *multi-armed bandit problems* by allowing job/bandit evolution while passive. We shall deploy Whittle's approach to the development of index policies for restless bandit problems to construct a simple, strongly performing class of fixed-time scheduling heuristics.

We firstly show how an individual job (with associated r, $p(\cdot)$ and T) may be formulated as a *restless bandit*.

(i) Identify the job's state space as $\{0, 1, \ldots, T\} \cup \{*\}$. State $s \in \{0, 1, \ldots, T-1\}$ is the amount of the job's past processing, job state T indicates that no further rewards may be earned from it, while if the job is in state $*$, it is ineligible for further processing;

(ii) Two actions $\{a, b\}$ may be applied to the job. Action a is *active* and indicates that processing is to be applied. Action a is admissible in states $\{0, 1, \ldots, T\}$ but not in state $\{*\}$. Action b is *passive* and indicates that processing is *not* to be applied. Action b is admissible in all states.

(iii) If action a is applied to the job at some time t in state $s \in \{0, 1, \ldots, T-1\}$ then the job's state at $t+1$ will be $s+1$. This transition earns an expected (discounted) reward $\beta r p(s+1)$ where $\beta \in (0, 1)$. If action a is applied when the job is in state T, the job's state does not change and no reward is earned. Under the passive action b we have the transitions $0 \to 0$, $* \to *$ and $s \to *$, $1 \leq s \leq T$. It is the latter transitions which induce the job's *restlessness* and which capture the structure of the fixed-time scheduling problem. Transitions under the passive action earn nothing.

We now follow Whittle (1988) in formulating a *W-subsidy problem* for the bandit in (i)–(iii) above. To develop this we first modify (iii) above so that all transitions under the passive action earn a (discounted) reward $W \in \mathbb{R}$. The W-subsidy problem is then the Markov Decision Problem in which at each time $t \in \mathbb{N}$ an action for the job is chosen from the admissible set to maximise the total expected discounted reward earned over an infinite horizon. The theory of stochastic dynamic programming (see, for example, Puterman (1994)) guarantees the existence of an optimal policy which is *stationary* and whose value function satisfies the optimality equations of dynamic programming (DP).

We write $V(\cdot, W)$ for the maximum expected return for the W-subsidy problem from initial state $\cdot$. From (i)–(iii) it satisfies the following DP equations:

$$V(0, W) = \max\{\beta r p(1) + \beta V(1, W);\ W + \beta V(0, W)\}, \tag{42}$$

$$V(s, W) = \max\{\beta r p(s+1) + \beta V(s+1, W);\ W + \beta V(*, W)\},\ 1 \leq s \leq T-1, \tag{43}$$

$$V(T, W) = \max\{\beta V(T, W);\ W + \beta V(*, W)\}, \tag{44}$$

$$V(*, W) = W(1-\beta)^{-1}. \tag{45}$$

However, substituting from (45) into (43), (44) and exploiting the appearance of $V(0, W)$ on both sides of (42) we see that (42)–(45) is equivalent to the following:

$$V(0, W) = \max\left\{\beta r p(1) + \beta V(1, W); W(1-\beta)^{-1}\right\}, \tag{46}$$

$$\begin{aligned} V(s, W) = \max\{&\beta r p(s+1) + \beta V(s+1, W); \\ &W(1-\beta)^{-1}\}, \ 1 \le s \le T-1, \end{aligned} \tag{47}$$

$$V(T, W) = \max\left\{0, W(1-\beta)^{-1}\right\}, \tag{48}$$

$$V(*, W) = W(1-\beta)^{-1}. \tag{49}$$

From (46)–(49) it is clear that the W-subsidy problem is equivalent to a *retirement problem* in which each of the states $s \in \{0, 1, \ldots, T-1\}$ a choice is made between allocating a unit in processing to the job (action a) or retiring from further processing (action b) and claiming a retirement reward of $W(1-\beta)^{-1}$. In state T, action a will be optimal if $W \le 0$ with b optimal for $W \ge 0$. Whittle (1980) studied such retirement problems and characterised optimal retirement policies.

We write $\Pi(W)$ for the set of states in which action b (passive, retire) is optimal for the W-subsidy problem. From Whittle (1980) and the above discussion we are able to infer that

$$\Pi(W) = \begin{cases} \{*\}, & W < 0, \\ \{T, *\} \cup \{s; 0 \le s \le T-1 \text{ and } G(s) \le W(1-\beta)^{-1}\}, & W \ge 0. \end{cases} \tag{50}$$

We now utilise the above solutions to the W-subsidy problem to develop a calibrating index for the job/restless bandit in (i)-(iii) by following Whittle (1988) as follows:

Definition 13 *The job in (i)-(iii) is* ***indexable*** *if $\Pi(W)$ is increasing in W. If it is indexable then the* ***Whittle index*** *$W : \{0, 1, \ldots, T\} \to \mathbb{R}^+$ is defined by*

$$W(s) = \inf\{W; s \in \Pi(W)\}, \ 0 \le s \le T. \tag{51}$$

The following is an immediate consequence of (50) and the above definition.

Theorem 14 *The job in (i)-(iii) is indexable with Whittle index given by*

$$W(s) = \begin{cases} (1-\beta)G(s), & 0 \le s \le T-1, \\ 0, & s = T. \end{cases} \tag{52}$$

Now consider the full fixed-time scheduling problem of Section 2 with job set $J = \{1, 2, \ldots, N\}$ and each job $j \in J$ having Whittle index $W_j(s) = (1-\beta)G_j(s)$, $0 \le s \le T$. Whittle's proposed heuristic will always choose to process a job with maximum Whittle index. The fixed-time schedules which result can be assumed to have the following structure:

1. Renumber the jobs such that

$$G_1(0) \ge G_2(0) \ge \ldots \ge G_N(0);$$

Table 3 Average percentage suboptimalities when two heuristics are applied to four problem types

Problem Type	A	B	C	D
Whittle index heuristic (H_1)	0.165	0.182	0.148	0.066
Improved Whittle index heuristic (H_2)	0.044	0.067	0.012	0.001

2. Process the jobs in numerical order with job j having allocated processing time $\tilde{t}_j$, where

$$\tilde{t}_j = \min\{t; G_j(t) < G_{j+1}(0)\},\ 1 \leq j \leq N-1, \tag{53}$$

and $\tilde{t}_N = T_N$. Note that, from (53) and Definition 1 it will follow that $\tilde{t}_j \in Min(j)$, $1 \leq j \leq N$.

Punton (2004) has further developed a systematic approach to the exploration of possible ways of improving the above Whittle index fixed-time scheduling heuristic. At each stage of the improvement algorithm, one of two different kinds of manoeuvres are considered, namely:

I. modification of the job ordering by a pairwise interchange (with allocated processing times unchanged);
II. replacement of the allocated processing for some job j by an alternative member of $Min(j)$ (with the job ordering unchanged).

Numerical experience

To illustrate the ideas of this section we report on the performance of the above heuristics for 400 randomly generated problems, each of which has $N = 10$. All jobs j have $T_j = 10$. There were four problem types (labelled A, B, C, D) with 100 problems generated for each. Problems of type A are such that all jobs earn a unit reward when completed with the probabilities $p_j(s)$, $1 \leq s \leq 10$, obtained by sampling from a uniform $U(2, 5)$ distribution and normalising. Jobs in problems of type B also earn a unit reward but probabilities $p_j(s)$, $1 \leq s \leq 10$, are obtained by sampling from $U(0, 5)$ and normalising (and hence will be more variable than those of type A). Jobs in problems of types C and D have probability characteristics which follow those of types A and B respectively. However, job rewards are now obtained by sampling from a $U(0.5, 1.5)$ distribution. In all cases we take $\beta = e^{-0.05}$.

For each member of each collection of 100 generated problems, the maximum reward V^{opt} from the class of fixed-time schedules was computed along with the rewards V^{H_1} and V^{H_2} from the two heuristics under investigation. Heuristic H_1 is the Whittle index fixed-time scheduling heuristic (see 1. and 2. above) while H_2 results from applying the improvement algorithm to H_1 (see I. and II. above). In all cases the percentage suboptimality $100\{(V^{opt} - V^{H_i})/V^{opt}\}$ of H_i was calculated $i = 1, 2$ and averaged over the 100 generated within each problem type. The results are presented in Table 3. It is plain that both heuristics perform very well with the improved Whittle index heuristic uniformly very close to optimal.

Acknowledgements The first author acknowledges the support of the Engineering and Physical Sciences Research Council (EPSRC) through the award of grant GR/S45188/01. The second author acknowledges EPSRC support provided through a research studentship.

References

Benkherouf, L., Glazebrook, K.D., Owen, R.W.: Single visit policies for allocating a single resource in a stochastic environment. Oper. Res. **42** 1087–1099 (1994)
Dacre, M.J., Glazebrook, K.D., Niño-Mora, J.: The achievable region approach to the optimal control of stochastic systems (with discussion). J. Roy. Statist. Soc. **B61**, 747–791 (1999)
Gaver, D.P., Jacobs, P.A., Samorodnitsky, G.: Modelling and analysis of uncertain time-critical tasking problems (UTCTP). Technical Report NPS-OR-03-005, Naval Postgraduate School, Monterey, CA (2003)
Gittins, J.C.: Multi-armed bandit allocation indices. Wiley, Chichester, 1989
Glazebrook, K.D.: Stochastic scheduling and forwards induction. Discrete Appl. Math. **57**, 145–165 (1995)
Papadimitriou, C.H., Tsitsiklis, J.N.: The complexity of optimal queueing network control. Math. Oper. Res. **24**, 293–305 (1999)
Punton, E.L.: Internal report. Edinburgh University, 2004
Puterman, M.L.: Markov decision processes: discrete stochastic dynamic programming. Wiley, New York, 1994
Rieder, U., Wagner, H.: Structured policies in the sequential design of experiments. Ann. Oper. Res. **32**, 189–203 (1991)
Whittle, P.: Multi-armed bandits and the Gittins index. J. Roy. Statist. Soc. **B42**, 143–149 (1980)
Whittle, P.: Restless bandits: activity allocation in a changing world. J. Appl. Prob. **A25**, 287–298 (1988)

Appendix B

The following is a paper forthcoming in Naval Research Logistics. It contains material covered in Chapter 3 of the thesis.

Dynamic Policies for Uncertain Time-Critical Tasking Problems

K.D. Glazebrook[1] and E.L. Punton[2]

[1]Department of Mathematics and Statistics and the Management School, Lancaster University, Lancaster, LA1 4YX, United Kingdom

[2]School of Management, University of Edinburgh, Edinburgh, EH8 9JY, United Kingdom

Abstract

A recent paper by Gaver *et al.* (2006) argued the importance of studying service control problems in which the usual assumptions (i) that tasks will wait indefinitely for service and (ii) that successful service completions can be observed instantaneously are relaxed. Military and other applications were cited. They proposed a model in which arriving tasks are available for service for a period whose duration is unknown to the system's controller. The allocation of a large amount of processing to a task may make more likely its own successful completion but may also result in the loss of many unserved tasks from the system. Gaver *et al.* (2006) called for the design of dynamic policies for the allocation of service which maximise the rate of successful task completions achieved, or which come close to doing so. This is the theme of the paper. We utilise dynamic programming policy improvement approaches to design heuristic dynamic policies for service allocation which may be easily computed. In all cases studied these policies achieve throughputs close to optimal.

1 Introduction

The paper considers a simple scenario in which tasks arrive and seek service which is provided by a single server. In most standard models of such systems, strong assumptions are made about (i) all tasks' limitless availability for service, namely their preparedness to wait in the system until service is delivered, and (ii) the server's capacity to deliver successful processing of all tasks and to know when this has been achieved. In a recent paper, Gaver *et al.* (2006) argued that in many situations in which service is offered, one or both of these assumptions may not hold.

As an example Gaver, Jacobs and Sato (2006) consider a Homeland Security scenario in which hostile Red vessels arrive at and move through a maritime domain towards a vulnerable target, protected by Blue. The domain is also visited by non-hostile vessels

(Whites) of interest. In this context service by Blue consists of an attempted classification (into non-hostile White or hostile Red) of suspicious vessels encountered in the domain by an overhead sensor. The Blue sensor will track any vessel it classifies as Red until it is relieved by another platform, perhaps a destroyer pair. While correct classification by the sensor is never certain in finite time, the longer the sensor spends on classification the more likely it is to be correct. However, long services by Blue increase the chance that hostile Reds will traverse the domain to the target (i.e. be lost from the system) unclassified. The question naturally arises as to how the sensor should be deployed (i.e., how long allocated times spent on classification should be) to achieve a maximised correct classification rate among all vessels arriving at the domain.

Gaver *et al.* (2006) mention other scenarios in which Red agents seek to penetrate Blue defenses. In one such, a Red will leave once his offensive task has been completed while Blue's service of Red seeks his destruction, with longer services more likely to achieve this. In such a scenario, Blue may not be able to establish with certainty whether any attempt to destroy Red has been successful. Gaver *et al.* (2006) also cite examples from the delivery of emergency medical treatment.

We shall consider a scenario in which tasks arrive at a single server according to a Poisson process. Each task has a limited availability for useful service which is unknown to the controller of the system. Once this time has expired, the task is lost. The case of tasks with *known* deadlines is important and (relatively) well studied. See, for example, Glazebrook (1983), Jiang, Lewis and Colin (1996), Lehoczky (1996, 1997a, 1997b) and Doytchinov, Lehoczky and Shreeve (2001). We shall further suppose that the server has limited information concerning the efficacy of each service. In light of this, the server adopts an approach to processing in which a period of allocated time for the service of a given task is determined in advance to its processing. In choosing such *allocated service times* there is a clear trade-off to be considered. Large allocated times are more likely to lead to a successful service, but may also lead to more waiting tasks being lost from the system. How this trade-off is resolved will plainly depend upon the number of currently waiting tasks at risk of loss. We would expect shorter allocated times to be appropriate when the task queue is long. Gaver *et al.* (2006) studied the *static* problem of determining the *constant* allocated service time which, when applied to *all* served tasks, maximises *throughput* (the time average rate of tasks served successfully) and found that even this simple version of the service allocation problem is extremely challenging. They further made a simple myopic proposal for the *dynamic* problem in which service allocations take account of queue length, but argued that the development of approximately optimal policies remained an important outstanding issue. It is this issue which is the concern of the current paper.

The authors know of no previous work on their model, save that of Gaver *et al.* (2006), cited above. In earlier work, Glazebrook and Punton (2005) considered a simpler set up in which the penalty imposed upon waiting tasks for large allocated service times is expressed through a discounted reward structure. Further, Glazebrook *et al.* (2004) and Harrison and Zeevi (2004) have explored the optimal dynamic allocation of service in the face of customer loss through impatience in contexts where the successful completion of service is observed immediately. There is also a developing literature concerning how the phenomenon of customer impatience should impact the design and operation of call centres. See, for example, Garnett, Mandelbaum and Reiman (2002) and Bassamboo, Harrison and Zeevi (2005).

The paper is structured as follows: our service control model is introduced in Section

2, while in Section 3 a tractable class of Markovian policies is introduced. These were first discussed by Gaver *et al.* (2006) and permit calculation of key performance measures, including throughput, via explicit formulae. In Section 4 we develop a class of dynamic heuristics (Heuristic I) by a dynamic programming (DP) policy improvement approach which applies a single DP step to the value function associated with the optimal Markovian policy. In Section 5, a second class of dynamic heuristics (Heuristic II) is developed by application of a single policy improvement step to a strongly performing static policy. The paper concludes in Section 6 with an account of a numerical investigation which testifies to the very strong performance of Heuristic II throughout. A carefully designed static policy, while certainly inferior to Heuristic II, is seen to perform remarkably well in the dynamic class quite widely, but does less well for problems where the achievable throughput from any policy is low.

2 The Model

Tasks seeking service arrive in a Poisson stream with rate λ. Once in the system, each task has an exponentially distributed amount of time (with mean θ^{-1}) during which it is available for service. These times are independent for distinct tasks. If a task's $\exp(\theta)$ *availability time* expires before successful service has been achieved, it leaves the system unprocessed. We shall suppose throughout that $\theta > 0$.

A single server is available to serve tasks. Decision epochs for the server are the times at which periods of allocated service expire (and the system is non-empty) together with the times at which tasks arrive at an empty system. At each decision epoch, the server observes the number of tasks currently present (n) and chooses a period of *allocated service* ($t(n)$) for one of them. The server is then committed to this task for this allocated period. Each task is assumed to have a *service requirement*, which is an unobservable positive-valued random variable with distribution function F. Service requirements for distinct tasks are independent and identically distributed. The probability that an allocated service of time t sees a task successfully completed is given by

$$\gamma(t) \equiv \int_0^t e^{-\theta s} dF(s), \tag{1}$$

namely the probability that the task's service requirement is less than $\min(t, S)$ where $S \sim \exp(\theta)$. Moreover, if n tasks are present at the beginning of this allocated service of length t (including the one about to be served) then it is easy to show that the number of tasks remaining at its conclusion is

$$X(t|n) + Y(t), \tag{2}$$

where $X(t|n)$ and $Y(t)$ are independent random variables taking values in the non-negative integers. The random variable $X(t|n)$ has a binomial distribution $Bin(n-1, e^{-\theta t})$ and is the number of those tasks present in the system at the start of the allocated service (excepting the one chosen for service) which remain at its conclusion. The random variable $Y(t)$ has a Poisson distribution $Poisson\{\lambda\theta^{-1}(1 - e^{-\theta t})\}$ and is the number of tasks which arrive in the system during the allocated service and which remain at its conclusion. The random variable $X(t|n)$ decreases stochastically in t for each n while $Y(t)$ increases stochastically in t. Whether the system is likely to experience a net gain or a net loss of tasks during the allocated service of length t depends critically upon the relative sizes of $n - 1$ and $\lambda\theta^{-1}$.

Each task is deemed to be unavailable for further service at the conclusion of its allocated service time.

A *stationary policy* is given by a map $t(\cdot) : \mathbb{Z}^+ \to \mathbb{R}^+$ which determines service times as a function of the number of tasks present. The goal of analysis is the determination of a stationary policy which maximises throughput, or which comes close to doing so. However, this is a stochastic dynamic optimization problem which is challenging in several respects. First, its state space is countably infinite (the natural numbers $\mathbb{N}$) while its action space is uncountable (the positive reals $\mathbb{R}^+$). Second, actions taken relate directly to the expiration of time which means that many standard approaches require modification. A conventional proposal would apply stochastic DP to a series of finite state/finite action/discrete time approximations to the system. Each would involve truncating the state space (replacing $\mathbb{N}$ by $\{0, 1, \ldots, N\}$ for some suitably large N) and introducing a positive discrete time quantum δ along with a truncated finite action space ($\{\delta, 2\delta, \ldots, M\delta\}$ for some suitably large M). The existence of an optimal stationary policy $t(\cdot; N, M, \delta)$ is guaranteed for any choice of (N, M, δ). However, its determination by conventional DP is computationally expensive for large N, M and small δ. Moreover, the need to demonstrate *convergence* of the solution (in relation to throughputs achieved), namely that suitably increasing N, M while reducing δ will have negligible benefits, is exacting.

The paper presents two classes of effective dynamic heuristics for this service control problem, whose design avoids the above complex and cumbersome solution procedures. These policies are easy to compute. Moreover, numerical evidence suggests that they achieve throughputs which are close to optimal in all cases studied. Both policy classes emerge from a simple *Markovian* policy class which is discussed in Gaver *et al.* (2006) and which is described briefly in the next section.

Comment

For simplicity and clarity we shall focus exclusively on the scenario described in the opening two paragraphs of this section. However, the methods of analysis we describe in the upcoming sections are readily extended to situations in which (a) tasks are guaranteed not to depart the system during service, and/or (b) a delivered service of length t achieves service quality or customer satisfaction $s(t) \in \mathbb{R}^+$, with $s(t)$ increasing in t. In the latter case, the goal of analysis is the determination of a stationary policy which maximises the time average level of customer satisfaction achieved over an infinite horizon, or which comes close to doing so.

Regarding (b) above, please note that Deshmukh and Jain (1977) consider a finite capacity queueing model in which the objective is maximisation of customer satisfaction net of (linear) waiting costs. There are no losses from queue or from service. In their simple finite state scenario they are able to obtain characterisations of optimal policies. They do not discuss the computation of optimal or near optimal policies.

3 A static Markovian policy class

Suppose that, in the context of the problem described in Section 2, every allocated service is drawn independently from an exponential distribution with mean μ^{-1}. It then follows that, in advance of sampling, each allocated service results in a successful service completion with

probability

$$\bar{\gamma}(\mu) \equiv \int_0^\infty e^{-(\theta+\mu)t} dF(t). \tag{3}$$

Moreover, the number of tasks present in the system (namely the task currently being served together with any waiting for service) follows a state dependent birth-death process with birth (arrival) rate λ in all states and death (completion or loss) rate $\mu + \theta(n-1)$ in state $n \geq 1$.

Standard theory indicates that the throughput achieved under the above policy is given by

$$\bar{\omega}(\mu) = \mu\bar{\gamma}(\mu) \left[\sum_{n=1}^{\infty} \left\{ \prod_{k=0}^{n-1} \lambda(\mu + \theta k)^{-1} \right\} \right] \left[1 + \sum_{n=1}^{\infty} \left\{ \prod_{k=0}^{n-1} \lambda(\mu + \theta k)^{-1} \right\} \right]^{-1}. \tag{4}$$

Gaver *et al.* (2006) argue that $\bar{\omega}(\mu)$ is maximised by some finite μ, μ^* say. Numerical evidence is given in Gaver *et al.* (2006) that the policy of giving each served task an allocation fixed in size and equal to $(\mu^*)^{-1}$ is close to optimal in the *static class* of policies in which all allocated services are equal.

Comment

All of the numerical examples considered in the paper suppose that service times are drawn independently from a gamma distribution $\Gamma(r, \nu)$ with mean r/ν and probability density function

$$f(t|r, \nu) = \nu^r t^{r-1} e^{-\nu t} \{\Gamma(r)\}^{-1}, \ t \geq 0. \tag{5}$$

In this case the quantity $\bar{\gamma}$ in (3) is given by

$$\bar{\gamma}(\mu) = \{\nu/(\nu + \theta + \mu)\}^r. \tag{6}$$

It is standard that such service times are stochastically increasing in r (with ν fixed) and stochastically decreasing in ν (with r fixed). It is straightforward to show that, for any fixed μ, the throughput $\bar{\omega}(\mu)$ is increasing in ν and λ, while decreasing in r and θ. In all cases, the non-specified parameters are assumed to be held constant. Further, the quantity $\bar{\omega}(\mu)/\lambda$, namely the proportion of arriving tasks which are successfully served, is decreasing in λ for any fixed μ.

To facilitate the analysis of Section 4 we shall require the *bias function* associated with the Markov policy μ, written $b(\cdot, \mu) : \mathbb{N} \to \mathbb{R}$ and which measures the relative transient effect of commencing processing in general state n rather than 0. To develop this, we require the following quantities:

- for T, $\mu \in \mathbb{R}^+$ and $n \in \mathbb{N}$ we write $C(n, T; \mu)$ for the expected number of successful task completions during the period $[0, T)$ under Markov policy μ when n is the system state at time 0;
- for $\mu \in \mathbb{R}^+$ and $n \in \mathbb{Z}^+$ we write $\bar{C}(n, \mu)$ for the expected number of successful task completions under Markov policy μ from time 0 up to the time at which the system empties for the first time when n is the initial system state;

- for $\mu \in \mathbb{R}^+$ and $n \in \mathbb{Z}^+$ we write $\bar{T}(n,\mu)$ for the expected time it takes the system to empty for the first time from initial state n under Markov policy μ.

We pause to note that the system evolving under Markov policy μ is trivially ergodic, from which we deduce that the quantities $\bar{T}(n,\mu)$ and $\bar{C}(n,\mu)$ are guaranteed to be finite for any $n \in \mathbb{N}$. We now define the bias function $b(\cdot,\mu)$, which is also guaranteed to be finite by the above ergodicity and standard theory. See, for example, Tijms (1994).

Definition 1 *The bias function $b(\cdot,\mu) : \mathbb{N} \to \mathbb{R}$ is defined by*

$$b(n,\mu) = \lim_{T\to\infty} \{C(n,T;\mu) - C(0,T;\mu)\}, \tag{7}$$

where the limit in (7) is guaranteed to exist and be finite for any $n \in \mathbb{N}$.

Lemma 1 *The bias $b(n,\mu)$ is given by*

$$b(n,\mu) = \frac{\mu\bar{\gamma}(\mu)}{\lambda}\left[\sum_{m=1}^{n}\sum_{r=1}^{\infty}\left\{\prod_{k=0}^{r-1}\lambda(\mu+\theta(m-1)+\theta k)^{-1}\right\}\right]\left[1+\sum_{r=1}^{\infty}\left\{\prod_{k=0}^{r-1}\lambda(\mu+\theta k)^{-1}\right\}\right]^{-1},$$

$$n \in \mathbb{Z}^+,$$

and is zero when $n = 0$.

Proof By deployment of the ergodicity of the system under Markov policy μ and utilising the fact that the system regenerates upon entry into the empty state, standard theory (Tijms (1994)) yields that

$$b(n,\mu) = \bar{C}(n,\mu) - \bar{\omega}(\mu)\bar{T}(n,\mu),\ n \in \mathbb{Z}^+. \tag{8}$$

Plainly from (7) we have that $b(0,\mu) = 0$. The quantity $\bar{\omega}(\mu)$ is given by the expression in (4).

In order to compute $\bar{C}(n,\mu)$ and $\bar{T}(n,\mu)$, we firstly consider the quantity $\bar{T}(m,\mu) - \bar{T}(m-1,\mu)$ for $m \geq 1$. To compute this, we consider a state dependent birth-death process defined on the state space $\{m-1, m, m+1, \ldots\}$. The birth rate in all states is λ, while the death rate in state $m-1$ is zero and in all other states is $\mu + \theta(n-1)$, $n \geq m$. By direct comparison with our service system evolving under Markov policy μ we see that the stationary probability that the above birth-death process is in its lowest state $(m-1)$ may be expressed as

$$\lambda^{-1}\left\{\lambda^{-1} + \bar{T}(m,\mu) - \bar{T}(m-1,\mu)\right\}^{-1}. \tag{9}$$

However, by standard theory, this probability may also be expressed as

$$\left[1+\sum_{r=1}^{\infty}\left\{\prod_{k=0}^{r-1}\lambda(\mu+\theta(m-1)+\theta k)^{-1}\right\}\right]^{-1}. \tag{10}$$

Equating the expressions in (9) and (10) yields the relation

$$\bar{T}(m,\mu) - \bar{T}(m-1,\mu) = \lambda^{-1} \sum_{r=1}^{\infty} \left\{ \prod_{k=0}^{r-1} \lambda(\mu + \theta(m-1) + \theta k)^{-1} \right\}, \ m \in \mathbb{Z}^+. \tag{11}$$

It is straightforward to establish that

$$\bar{C}(m,\mu) - \bar{C}(m-1,\mu) = \mu\bar{\gamma}(\mu) \left\{ \bar{T}(m,\mu) - \bar{T}(m-1,\mu) \right\}, \ m \in \mathbb{Z}^+. \tag{12}$$

We now obtain the expression for $b(n,\mu)$ in the lemma by writing

$$\begin{aligned} b(n,\mu) &= \bar{C}(n,\mu) - \bar{\omega}(\mu)\bar{T}(n,\mu) \\ &= \sum_{m=1}^{n} \left[\bar{C}(m,\mu) - \bar{C}(m-1,\mu) - \bar{\omega}(\mu) \left\{ \bar{T}(m,\mu) - \bar{T}(m-1,\mu) \right\} \right] \\ &= \left\{ \mu\bar{\gamma}(\mu) - \bar{\omega}(\mu) \right\} \sum_{m=1}^{n} \left\{ \bar{T}(m,\mu) - \bar{T}(m-1,\mu) \right\} \end{aligned}$$

and using (4) and (11). This concludes the proof. □

4 Heuristic I – dynamic policy development from the Markovian class

We use the notation $\{t(\cdot), \mu^*\}$ to denote a policy which chooses its *first* allocated service according to the map $t(\cdot) : \mathbb{Z}^+ \to \mathbb{R}^+$ and which makes all subsequent decisions according to optimal Markovian policy μ^*. In a sense to be made precise below our first heuristic is given by a map $\bar{t}(\cdot) : \mathbb{Z}^+ \to \mathbb{R}^+$ enjoying the property that $\{\bar{t}(\cdot), \mu^*\}$ achieves the maximum expected number of successful task completions over an infinite horizon among the class of policies $\{t(\cdot), \mu^*\}$, uniformly over all initial states of the system. We may assume without loss of generality in what follows that the system is non-empty at time zero.

Developing the notation of Section 3 we write $C(n,T;t,\mu)$ for the expected number of successful task completions during the period $[0,T)$ under the policy (t,μ) whose first allocated service is t and which makes all subsequent decisions according to Markov policy μ when n is the system state at time 0. For definiteness, we suppose here and hereafter that $T \geq t$.

Lemma 2 *The difference between the expected number of successful task completions achieved over an infinite horizon by policies (t,μ) and μ from initial state $n \in \mathbb{Z}^+$ is given by*

$$\lim_{T\to\infty} \left\{ C(n,T;t,\mu) - C(n,T;\mu) \right\} = \gamma(t) + \mathbb{E}\left[b\left\{ X(t|n) + Y(t), \mu \right\} \right] - b(n,\mu) - t\bar{\omega}(\mu), \tag{13}$$

where in (13), $X(t|n) \sim Bin(n-1, e^{-\theta t})$ and $Y(t) \sim Poisson\left\{ \lambda\theta^{-1}(1-e^{-\theta t}) \right\}$ are independent random variables.

Proof If the first allocated service is t and the initial system state is $n \in \mathbb{Z}^+$ then by conditioning on the system state at time t we obtain

$$C(n,T;t,\mu) = \gamma(t) + \mathbb{E}\left[C\left\{ X(t|n) + Y(t), T-t; \mu \right\} \right], \tag{14}$$

where $X(t|n)$ and $Y(t)$ are as in the statement of the lemma. We now expand (14) by writing

$$C(n,T;t,\mu) - C(n,T;\mu) = \gamma(t) + \mathbb{E}\left[C\left\{X(t|n) + Y(t), T-t;\mu\right\} - C(0,T-t;\mu)\right]$$

$$-\{C(0,T;\mu) - C(0,T-t;\mu)\} - \{C(n,T;\mu) - C(0,T;\mu)\}. \quad (15)$$

We now take the limit $T \to \infty$ on both sides of (15). First note that it is easily deduced from Lemma 1, that the bias function satisfies

$$0 \leq b(n,\mu) \leq n\bar{\gamma}(\mu), \ n \in \mathbb{N}. \quad (16)$$

Further, it is straightforward that

$$C(n,T;\mu) - C(0,T;\mu) \leq \bar{C}(n,\mu) = \left\{1 - \frac{\bar{\omega}(\mu)}{\mu\bar{\gamma}(\mu)}\right\}^{-1} b(n,\mu), \ T \in \mathbb{R}^+, \ n \in \mathbb{N}. \quad (17)$$

By utilising (16) and (17) together with Definition 1, we infer that

$$\lim_{T\to\infty} \mathbb{E}\left[C\{X(t|n) + Y(t), T-t;\mu\} - C(0,T-t;\mu)\right] = \mathbb{E}\left[b\{X(t|n) + Y(t),\mu\}\right]. \quad (18)$$

By application of Blackwell's Theorem to the renewal process consisting of successive entries of the system (evolving under Markov policy μ) into the empty state, we conclude that

$$\lim_{T\to\infty} \{C(0,T;\mu) - C(0,T-t;\mu)\} = t\bar{\omega}(\mu). \quad (19)$$

The result now follows from (14), (15), (18), (19) and Definition 1. This concludes the proof. □

We now develop the map $\bar{t}(\cdot) : \mathbb{Z}^+ \to \mathbb{R}^+$ by choosing

$$\bar{t}(n) = \underset{t\geq 0}{\operatorname{argmax}} \left(\gamma(t) + \mathbb{E}\left[b\{X(t|n) + Y(t);\mu^*\}\right] - t\bar{\omega}(\mu^*)\right), \quad (20)$$

where $X(t|n)$ and $Y(t)$ are as in the statement of Lemma 2. If we suppose that the service requirement is absolutely continuous then the expression on the r.h.s. of (20) is certainly continuous. Moreover, since both $\gamma(t)$ and $\mathbb{E}[b\{X(t|n) + Y(t);\mu^*\}]$ can easily be shown to converge to finite limits as $t \to \infty$, this expression must be decreasing in t beyond some finite value T_n, say. Hence maxima must occur in $[0, T_n]$. It now follows from standard real analysis that the maximum in (20) must be achieved. For definiteness choose $\bar{t}(n)$ to be the largest value achieving the maximum. The following is an immediate consequence of Lemma 2 and the construction of $\bar{t}(\cdot)$.

Theorem 3 *The map $\bar{t}(\cdot) : \mathbb{Z}^+ \to \mathbb{R}^+$ is such that*

$$\lim_{T\to\infty} \left[C\{n,T;\bar{t}(n),\mu^*\} - C\{n,T;t(n),\mu^*\}\right] \geq 0, \ n \in \mathbb{Z}^+,$$

for any choice of $t(\cdot) : \mathbb{Z}^+ \to \mathbb{R}^+$.

Theorem 3 substantiates the claim made in the opening paragraph of this section, namely that $\{\bar{t}(\cdot),\mu^*\}$ maximises the expected number of successful task completions among the policy class $\{t(\cdot),\mu^*\}$ uniformly over all initial states. Heuristic I is given by the map $\bar{t}(\cdot) : \mathbb{Z}^+ \to \mathbb{R}^+$, namely it chooses allocated service $\bar{t}(n)$ when the queue length is n.

(λ, ν)	θ \ n	$r = 2$								$r = 1$							
		1	2	3	4	5	6	8	10	1	2	3	4	5	6	8	10
$(0.25, 0.3)$	0.1	7.98	7.37	6.95	6.68	6.51	6.39	6.26	6.19	4.73	3.78	3.23	2.92	2.73	2.61	2.47	2.39
	0.2	6.59	6.10	5.71	5.44	5.26	5.15	5.03	4.99	3.85	2.97	2.47	2.23	2.10	2.03	1.96	1.92
	0.3	5.67	5.26	4.90	4.63	4.43	4.31	4.19	4.15	3.32	2.51	2.03	1.82	1.73	1.69	1.65	1.63
$(0.9, 0.3)$	0.1	5.78	5.67	5.59	5.54	5.50	5.48	5.45	5.43	2.72	2.33	2.04	1.84	1.69	1.58	1.45	1.37
	0.2	4.86	4.74	4.65	4.58	4.53	4.50	4.45	4.43	2.30	1.94	1.68	1.51	1.40	1.32	1.23	1.18
	0.3	4.21	4.10	4.00	3.93	3.88	3.83	3.78	3.75	2.03	1.70	1.46	1.31	1.21	1.15	1.08	1.05
$(0.9, 0.8)$	0.1	3.23	3.03	2.89	2.79	2.73	2.68	2.65	2.62	1.95	1.64	1.44	1.30	1.20	1.14	1.04	0.99
	0.2	2.93	2.74	2.62	2.53	2.48	2.44	2.41	2.39	1.70	1.40	1.21	1.09	1.02	0.97	0.90	0.86
	0.3	2.70	2.53	2.41	2.33	2.28	2.24	2.19	2.17	1.54	1.25	1.07	0.97	0.90	0.86	0.81	0.78

(λ, ν)	θ \ n	$r = 0.5$								$r = 0.25$							
		1	2	3	4	5	6	8	10	1	2	3	4	5	6	8	10
$(0.25, 0.3)$	0.1	3.13	2.09	1.60	1.36	1.22	1.13	1.03	0.97	2.22	1.21	0.85	0.68	0.59	0.53	0.47	0.43
	0.2	2.49	1.49	1.11	0.95	0.87	0.82	0.77	0.74	1.73	0.79	0.54	0.44	0.39	0.36	0.33	0.31
	0.3	2.13	1.18	0.87	0.76	0.70	0.68	0.64	0.63	1.47	0.60	0.41	0.34	0.31	0.29	0.27	0.26
$(0.9, 0.3)$	0.1	1.68	1.27	0.99	0.82	0.71	0.64	0.55	0.50	1.17	0.78	0.56	0.44	0.37	0.32	0.27	0.24
	0.2	1.38	0.97	0.74	0.61	0.53	0.48	0.43	0.40	0.93	0.55	0.38	0.30	0.25	0.22	0.19	0.17
	0.3	1.20	0.81	0.61	0.50	0.44	0.41	0.37	0.34	0.80	0.44	0.30	0.23	0.20	0.18	0.16	0.14
$(0.9, 0.8)$	0.1	1.33	1.01	0.81	0.69	0.61	0.56	0.49	0.44	0.97	0.66	0.49	0.40	0.34	0.30	0.25	0.22
	0.2	1.12	0.80	0.62	0.53	0.47	0.43	0.38	0.35	0.80	0.49	0.35	0.28	0.24	0.21	0.18	0.16
	0.3	1.00	0.68	0.52	0.44	0.39	0.36	0.32	0.30	0.70	0.39	0.27	0.22	0.19	0.17	0.15	0.13

Table 1: Values of the allocated service $\bar{t}(n)$ determined by Heuristic I for $n = 1(1)6, 8, 10$ for a range of problems with gamma $\Gamma(r, \nu)$ service times

Numerical examples

Table 1 shows values of the allocated service times under Heuristic I, namely $\bar{t}(n)$, for a range of problems in which service times are assumed to be drawn independently from a gamma distribution $\Gamma(r, \nu)$ with probability density function given in (5) above. For the time being, expand the notation to $\bar{t}(n, \lambda, \theta, r, \nu)$ to express dependence upon the key model parameters. We would expect these allocated service times to *decrease* in (i) n, the number of waiting tasks (more tasks at risk of loss from the system); (ii) λ, the rate of arriving tasks (more alternative tasks to service); and (iii) θ, the loss rate (long allocated services risk more severe losses). However, we would expect the allocated service times to *increase* as the times for successful service *increase* stochastically. Reassuringly, the results in Table 1 confirm these expectations. Formally, we conjecture that $\bar{t}(n, \lambda, \theta, r, \nu)$ is *decreasing* in n, λ, θ and ν but is *increasing* in r. In all cases, the non-specified parameters are assumed to be held constant. Sadly, the complexity of the r.h.s. of (20) has rendered it impossible to confirm any of these conjectures mathematically.

One case where we can get a partial result and some insight concerns the nature of the n-dependence of $\bar{t}(n, \lambda, \theta, r, \nu)$ when the loss rate θ is small. The only term on the r.h.s. of (20) which depends upon n is the second one involving the bias function. It is straightforward to show that for $n \geq 1$,

$$\frac{\partial}{\partial t}\mathbb{E}\left[b\left\{X(t|n) + Y(t); \mu^*\right\}\right] = \lambda\mathbb{E}\left[b\left\{n + \bar{Y}(t); \mu^*\right\} - b\left\{n - 1 + \bar{Y}(t); \mu^*\right\}\right] + O(\theta), \quad (21)$$

where $\bar{Y}(t) \sim \text{Poisson}(\lambda t)$ and $O(\theta)$ denotes a quantity which, when divided by θ, remains bounded in the limit $\theta \to 0$. From the expression for the bias given in Lemma 1, it is straightforward to show that $b(n, \mu)$ is increasing and concave in n, for fixed μ. It follows that the first term on the r.h.s. of (21) is both decreasing in t (for fixed n) and decreasing in n (for fixed t). This in turn suggests via straightforward analysis that we have $\bar{t}(n+1, \lambda, \theta, r, \nu) \leq \bar{t}(n, \lambda, \theta, r, \nu)$ when θ is small enough.

Contracting the notation again, we finally note from Table 1 that $\bar{t}(n)$ appears to be convex in n, with significant reductions in allocated service as the queue length increases through small values (1,2,3) but with fairly rapid subsequent convergence to a limit.

5 Heuristic II – policy improvement from the static class

By extension of the notation established in Section 4, we use $\{t(\cdot), [\tau]\}$ to denote a policy which chooses its *first* allocated service according to the map $t(\cdot) : \mathbb{Z}^+ \to \mathbb{R}^+$, with all subsequent allocated services equal to $\tau \in \mathbb{R}^+$. In a sense to be made precise, our second heuristic will be given by a map $\hat{t}(\cdot) : \mathbb{Z}^+ \to \mathbb{R}^+$ enjoying the property that $\{\hat{t}(\cdot), [(\mu^*)^{-1}]\}$ achieves the maximum expected number of successful task completions over an infinite horizon among the class of policies $\{t(\cdot), [(\mu^*)^{-1}]\}$.

Comments

1. Gaver *et al.* (2006) gave numerical evidence that a policy which gives an allocated service of $(\mu^*)^{-1}$ to each task comes close to achieving maximum throughput in the

static class of policies.

2. What we are in fact achieving in this section is the development of a strongly performing dynamic policy for service allocation by the application of a single DP policy improvement step applied to a strongly performing static policy. This is broadly in the spirit of Krishnan (1987) who discussed the optimal routing of incoming customers to parallel queues.

To develop our analysis, we shall require the following notations:

- for $T,\ t,\ \tau \in \mathbb{R}^+$ and $n \in \mathbb{N}$ we write $C(n, T; t, [\tau])$ for the expected number of successful task completions during the period $[0, T)$ under the policy $(t, [\tau])$ when n is the system state at time 0;
- for $T,\ \tau \in \mathbb{R}^+$ and $n \in \mathbb{N}$ we write $C(n, T; [\tau])$ for the expected number of successful task completions during $[0, T)$ under the static policy $[\tau]$ which gives allocated service τ to all tasks when n is the system state at time 0;
- for $\tau \in \mathbb{R}^+$ and $n \in \mathbb{Z}^+$ we write $\bar{C}(n, [\tau])$ for the expected number of successful task completions under static policy $[\tau]$ from 0 up to the time at which the system empties for the first time when n is the initial system state;
- for $\tau \in \mathbb{R}^+$ and $n \in \mathbb{Z}^+$ we write $\bar{T}(n, [\tau])$ for the expected time it takes the system to empty for the first time from initial state n under static policy $[\tau]$;
- for $\tau \in \mathbb{R}^+$ we write $\bar{\omega}([\tau])$ for the throughput achieved under static policy $[\tau]$. Utilising results of Baccelli *et al.* (1984) we have that

$$\bar{\omega}([\tau]) = \lambda\gamma(\tau)\left(\sum_{n=0}^{\infty}(\lambda\tau)^n\left[\prod_{k=1}^{n}\{(1-e^{-k\theta\tau})(k\theta\tau)^{-1}\}\right]\right) \times\left(1+\sum_{n=0}^{\infty}(\lambda\tau)^n\left[\prod_{k=1}^{n}\{(1-e^{-k\theta\tau})(k\theta\tau)^{-1}\}\right]\right)^{-1}. \tag{22}$$

Comment

For the gamma service time examples of (5), we find that, for any fixed τ, the throughput $\bar{\omega}([\tau])$ is increasing in $\nu,\ \lambda$ and decreasing in $r,\ \theta$. Further, the quantity $\bar{\omega}([\tau])/\lambda$ is decreasing in λ. In all cases, the non-specified parameters are assumed to be held constant. Note that these properties correspond exactly with those of throughput $\bar{\omega}(\mu)$ described in Section 2.

As with the class of Markov policies, we are able to define a bias function $b(\cdot, [\tau])$ for the static class, guaranteed finite whenever $\theta > 0$.

Definition 2 *The bias function* $b(\cdot, [\tau]) : \mathbb{N} \to \mathbb{R}$ *is defined by*

$$b(n, [\tau]) = \lim_{T\to\infty}\{C(n, T; [\tau]) - C(0, T; [\tau])\}, \tag{23}$$

where the limit in (23) *is guaranteed to exist and be finite for any* $n \in \mathbb{N}$.

The following result makes use of the fact that the system regenerates upon every entry into the empty state. See Tijms (1994).

Lemma 4 *The bias $b(n,[\tau])$ is given by*

$$b(n,[\tau]) = \bar{C}(n,[\tau]) - \bar{\omega}([\tau])\bar{T}(n,[\tau]),\ n \in \mathbb{Z}^+,$$

and is zero when $n = 0$.

In contrast to the account in Section 3, a closed form expression is not available for the bias function of a static policy. We shall now develop a numerical approach to its computation. First, we state a preparatory lemma.

Lemma 5 *The quantities $\bar{C}(n,[\tau])$ and $\bar{T}(n,[\tau])$ are such that*

$$\bar{C}(n,[\tau]) = \tau^{-1}\gamma(\tau)\bar{T}(n,[\tau]) \leq \gamma(\tau)\left\{n + \lambda\theta^{-1}e^{\lambda\theta^{-1}}\right\},\ n \in \mathbb{Z}^+. \tag{24}$$

Proof The equality in (24) is trivial. Hence it is enough to show that

$$\bar{T}(n,[\tau]) \leq \tau\left\{n + \lambda\theta^{-1}e^{\lambda\theta^{-1}}\right\},\ n \in \mathbb{Z}^+. \tag{25}$$

Consider the system evolving under policy $[\tau]$ from initial state $n \in \mathbb{Z}^+$. Write $\tilde{T}(n,[\tau])$ for the time at which the system empties for the first time. This is a random variable whose expectation is $\bar{T}(n,[\tau])$. It is not difficult to see that

$$\tilde{T}(n,[\tau]) \leq_{ST} \tau\left(n + \sum_{m=1}^{\infty} Y_m I\left[\bigcap_{r=1}^{m-1}\{Y_r > 0\}\right]\right),\ n \in \mathbb{Z}^+, \tag{26}$$

where in (26), $\leq_{ST}$ denotes stochastic ordering, $I[\cdot]$ is the indicator function and $\{Y_m, m \in \mathbb{Z}^+\}$ is a collection of independent and identically distributed Poisson random variables with mean $\lambda\theta^{-1}$.

To understand (26), we develop stochastic bounds on the times taken for successive attempts to empty the system under policy $[\tau]$. The first attempt processes the n tasks present initially, and is of duration no greater than $n\tau$. This attempt will be successful if the number of tasks present at its conclusion is zero. However, this number is stochastically bounded above by $Y_1 \sim Poisson(\lambda\theta^{-1})$ as is clear from comments in the paragraph following (2). Repeat the above with Y_1 the number of tasks present initially, and so on. The interpretation of Y_m is as a stochastic bound on the number of tasks present following the m^{th} attempt to empty the system. Plainly

$$\tilde{T}(n,[\tau]) \leq_{ST} \tau\left(n + \sum_{m=1}^{N} Y_m\right),\ n \in \mathbb{Z}^+, \tag{27}$$

where

$$N = \min(r; Y_r = 0). \tag{28}$$

Clearly (26) and (27) are equivalent. It is trivial to show that (25) is obtained by taking expectations on both sides of (26) while exploiting the independence of the Y_m's. This concludes the proof. □

In order to develop our numerical approach to the determination of key quantities, we write

$$P_{nm}(t) = P\{X(t|n) + Y(t) = m\} =$$

$$\sum_{r=0}^{\min(n-1,m)} \frac{(n-1)!}{r!(n-1-r)!(m-r)!} \lambda^{m-r} \theta^{r-m} e^{-r\theta t} (1 - e^{-\theta t})^{n+m-1-2r} \exp\{\lambda\theta^{-1}(1 - e^{-\theta t})\}, \tag{29}$$

where $t \in \mathbb{R}^+$, $n \in \mathbb{Z}^+$ and $m \in \mathbb{N}$ in (27). Utilisation of a simple conditioning argument yields the recursion

$$\bar{C}(n, [\tau]) = \gamma(\tau) + \sum_{m=1}^{\infty} P_{nm}(\tau) \bar{C}(m, [\tau]), \ n \in \mathbb{Z}^+. \tag{30}$$

To compute the function $\bar{C}(\cdot, [\tau]) : \mathbb{Z}^+ \to \mathbb{R}^+$, we develop the sequence of functions $\{f^{(r)}(\cdot) : \mathbb{Z}^+ \to \mathbb{R}^+, r \in \mathbb{Z}^+\}$ defined recursively by

$$f^{(1)}(n) = 0, \ n \in \mathbb{Z}^+, \tag{31}$$

and

$$f^{(r+1)}(n) = \gamma(\tau) + \sum_{m=1}^{\infty} P_{nm}(\tau) f^{(r)}(m), \ n, \ r \in \mathbb{Z}^+. \tag{32}$$

Lemma 6 *The function sequence* $\{f^{(r)}(\cdot) : \mathbb{Z}^+ \to \mathbb{R}^+, r \in \mathbb{Z}^+\}$ *satisfies*

$$\lim_{r \to \infty} f^{(r)}(n) = \bar{C}(n, [\tau]), \ n \in \mathbb{Z}^+. \tag{33}$$

Proof A simple proof by induction uses (31) and (32) to establish that the sequence $\{f^{(r)}(n), r \in \mathbb{Z}^+\}$ is non-decreasing for each $n \in \mathbb{Z}^+$. A further induction, together with (30) and (31) establishes that

$$f^{(r)}(n) \leq \bar{C}(n, [\tau]), \ n \in \mathbb{Z}^+. \tag{34}$$

It must then follow that the limit on the l.h.s. of (33) must always exist.

We now define

$$\Delta^{(r)}(n) = \bar{C}(n, [\tau]) - f^{(r)}(n), \ n, \ r \in \mathbb{Z}^+. \tag{35}$$

By (34) the $\Delta^{(r)}(n)$ are all non-negative, and by (30), (31) and (32) satisfy the recursion

$$\Delta^{(1)}(n) = \bar{C}(n, [\tau]), \ n \in \mathbb{Z}^+, \tag{36}$$

and

$$\Delta^{(r+1)}(n) = \sum_{m=1}^{\infty} P_{nm}(\tau)\Delta^{(r)}(n),\ n,\ r \in \mathbb{Z}^+. \tag{37}$$

Denote by $\{X(t), t \in \mathbb{N}\}$ a Markov chain with state space $\mathbb{N}$ evolving according to the transition matrix $\mathbf{P}'(\tau)$ where

$$P'_{nm}(\tau) = \begin{cases} \delta_{0m}, & n = 0,\ m \in \mathbb{N}, \\ \\ P_{nm}(\tau), & n \in \mathbb{Z}^+,\ m \in \mathbb{N}, \end{cases} \tag{38}$$

with δ in (38) the Kronecker delta. Plainly, 0 is an absorbing state for the chain, with all other states transient. It is trivial to show that (36), (37) and (38) together with Lemma 5 yields

$$\begin{aligned} \Delta^{(r+1)}(n) &= \mathbb{E}(C\{X(r), [\tau]\}|X(0) = n) \\ \\ &\leq \gamma(\tau)\mathbb{E}\left\{X(r) + \lambda\theta^{-1}e^{\lambda\theta^{-1}}I[X(r) > 0]|X(0) = n\right\} \\ \\ &\to 0,\ r \to \infty,\ n \in \mathbb{Z}^+, \end{aligned} \tag{39}$$

where the limit in (39) is established by a simple modification of the argument of the proof of Lemma 5. The statement in (33) is an immediate consequence of (39). This concludes the proof. □

From Lemmas 4 and 5 we observe that

$$b(n, [\tau]) = \left[1 - \bar{\omega}([\tau])\tau\{\gamma(\tau)\}^{-1}\right]\bar{C}(n, [\tau]),\ n \in \mathbb{Z}^+. \tag{40}$$

Hence the recursive scheme of Lemma 6 together with the expression in (22) enable the bias function $b(\cdot, [\tau])$ to be computed.

We now describe how to deploy the bias function to affect the DP policy improvement step which is the basis for the design of Heuristic II. The proof of Lemma 7 is similar to that of Lemma 2 and is omitted.

Lemma 7 *The difference between the expected number of successful task completions achieved over an infinite horizon by policies $(t, [\tau])$ and $[\tau]$ from initial state $n \in \mathbb{Z}^+$ is given by*

$$\begin{aligned} \lim_{T\to\infty}\{C(n, T; t, [\tau]) - C(n, T; [\tau])\} = \\ \gamma(t) + \mathbb{E}[b\{X(t|n) + Y(t)\}, [\tau]] - b(n, [\tau]) - t\bar{\omega}([\tau]), \end{aligned} \tag{41}$$

where in (41), $X(t|n) \sim Bin(n-1, e^{-\theta t})$ and $Y(t) \sim Poisson\{\lambda\theta^{-1}(1 - e^{-\theta t})\}$ are independent random variables.

Following Lemma 7, we now develop the map $\hat{t}(\cdot) : \mathbb{Z}^+ \to \mathbb{R}^+$ by choosing

$$\hat{t}(n) = \underset{t\geq 0}{\operatorname{argmax}}\left(\gamma(t) + \mathbb{E}[b\{X(t|n) + Y(t), [(\mu^*)^{-1}]] - t\bar{\omega}([(\mu^*)^{-1}])\right), \tag{42}$$

with $X(t|n)$ and $Y(t)$ as above. For reasons similar to those following (20), the maximum in (42) must be achieved when service requirements are absolutely continuous. Theorem 8 is an immediate consequence of Lemma 7 and the construction of the map $\hat{t}(\cdot)$.

Theorem 8 *The map $\hat{t}(\cdot) : \mathbb{Z}^+ \to \mathbb{R}^+$ is such that*

$$\lim_{T\to\infty}[C\{n,T;\hat{t}(n),[(\mu^*)^{-1}]\} - C\{n,T;t(n),[(\mu^*)^{-1}]\}] \geq 0,\ n \in \mathbb{Z}^+,$$

for any choice of $t(\cdot) : \mathbb{Z}^+ \to \mathbb{R}^+$.

Theorem 8 substantiates the claim made in the opening paragraph of this section, namely that $\{\hat{t}(\cdot),[(\mu^*)^{-1}]\}$ maximises the expected number of successful task completions among the policy class $\{t(\cdot),[(\mu^*)^{-1}]\}$ uniformly over all initial states. Corollary 9 now follows. In its statement we use $C\{n,T;\hat{t}(\cdot)\}$ to denote the expected number of successful task completions during the period $[0,T)$ under the stationary policy $\hat{t}(\cdot) : \mathbb{Z}^+ \to \mathbb{R}^+$ when n is the system state at time 0.

Corollary 9 *The map $\hat{t}(\cdot) : \mathbb{Z}^+ \to \mathbb{R}^+$ is such that*

$$\lim_{T\to\infty}\left[C\{n,T;\hat{t}(\cdot)\} - C\{n,T;[(\mu^*)^{-1}]\}\right] \geq 0,\ n \in \mathbb{Z}^+.$$

Proof First, by substitution of the map $t(n) = (\mu^*)^{-1},\ n \in \mathbb{Z}^+$, into the statement of Theorem 8, we have that

$$\lim_{T\to\infty}\left[C\{n,T;\hat{t}(n),[(\mu^*)^{-1}]\} - C\{n,T;[(\mu^*)^{-1}]\}\right] \geq 0,\ n \in \mathbb{Z}^+,$$

and hence that the policy $\{\hat{t}(\cdot),[(\mu^*)^{-1}]\}$ achieves a higher expected number of successful task completions than does the static policy $[(\mu^*)^{-1}]$. We now use standard arguments to infer that a policy whose first r decisions are made according to the map $\hat{t}(\cdot) : \mathbb{Z}^+ \to \mathbb{R}^+$, with all remaining decisions made according to $[(\mu^*)^{-1}]$ will outperform $[(\mu^*)^{-1}]$ itself. To obtain the result we now take the limit $r \to \infty$. This concludes the proof. □

The map $\hat{t}(\cdot) : \mathbb{Z}^+ \to \mathbb{R}^+$ determines Heuristic II, namely that allocated service $\hat{t}(n)$ is chosen when the queue length is n. We see from Corollary 9 that Heuristic II's throughput is guaranteed to be at least as large as that of the static policy $[(\mu^*)^{-1}]$ from which it was developed.

Numerical examples

In Table 2 you find values of the map $\hat{t}(n)$ for the same range of problems as is considered in Table 1 at the conclusion of Section 4. The map $\hat{t}(\cdot) : \mathbb{Z}^+ \to \mathbb{R}^+$ has the same monotone characteristics as the map $\bar{t}(\cdot) : \mathbb{Z}^+ \to \mathbb{R}^+$ described earlier. In all cases we have $\hat{t}(n) \leq \bar{t}(n)$, namely that Heuristic II allocates smaller service times than does Heuristic I for any given queue length . To see why this might be so, compare the expressions on the r.h.s. of (20) and (42) for some fixed $n \geq 1$. The first term is $\gamma(t)$ in both cases. Suppose now that the bias terms can be assumed to be uniformly close to each other and hence that the comparison is

		$r = 2$								$r = 1$							
(λ, ν)	θ \ n	1	2	3	4	5	6	8	10	1	2	3	4	5	6	8	10
$(0.25, 0.3)$	0.1	6.79	5.99	5.52	5.26	5.10	5.00	4.87	4.79	4.01	2.89	2.40	2.15	2.01	1.92	1.81	1.74
	0.2	5.52	4.82	4.38	4.16	4.05	4.00	3.93	3.89	3.29	2.23	1.83	1.68	1.60	1.54	1.47	1.44
	0.3	4.79	4.18	3.73	3.50	3.40	3.37	3.34	3.32	2.86	1.86	1.51	1.42	1.37	1.33	1.28	1.25
$(0.9, 0.3)$	0.1	5.07	5.03	5.01	5.00	4.99	4.98	4.98	4.97	2.10	1.62	1.32	1.14	1.03	0.96	0.87	0.82
	0.2	4.04	3.96	3.91	3.88	3.86	3.85	3.83	3.83	1.81	1.36	1.10	0.97	0.89	0.84	0.78	0.75
	0.3	3.40	3.31	3.24	3.20	3.18	3.16	3.14	3.13	1.62	1.19	0.97	0.86	0.80	0.76	0.71	0.68
$(0.9, 0.8)$	0.1	2.67	2.41	2.28	2.20	2.16	2.13	2.10	2.08	1.60	1.25	1.04	0.92	0.84	0.79	0.72	0.68
	0.2	2.41	2.18	2.05	1.98	1.94	1.91	1.88	1.86	1.40	1.05	0.87	0.78	0.72	0.68	0.63	0.60
	0.3	2.23	2.00	1.88	1.81	1.77	1.74	1.71	1.70	1.28	0.94	0.77	0.69	0.64	0.61	0.57	0.55

		$r = 0.5$								$r = 0.25$							
(λ, ν)	θ \ n	1	2	3	4	5	6	8	10	1	2	3	4	5	6	8	10
$(0.25, 0.3)$	0.1	2.79	1.69	1.28	1.09	0.98	0.92	0.83	0.79	2.07	1.06	0.74	0.60	0.52	0.47	0.42	0.38
	0.2	2.23	1.18	0.90	0.79	0.73	0.69	0.64	0.61	1.61	0.68	0.48	0.40	0.35	0.33	0.30	0.28
	0.3	1.92	0.92	0.72	0.64	0.60	0.57	0.54	0.53	1.37	0.51	0.36	0.31	0.28	0.27	0.24	0.23
$(0.9, 0.3)$	0.1	1.41	0.97	0.72	0.59	0.51	0.46	0.40	0.37	1.06	0.66	0.47	0.36	0.31	0.27	0.23	0.20
	0.2	1.17	0.74	0.54	0.45	0.40	0.36	0.32	0.30	0.84	0.47	0.32	0.25	0.21	0.19	0.16	0.15
	0.3	1.03	0.61	0.45	0.38	0.34	0.31	0.28	0.27	0.73	0.37	0.25	0.20	0.17	0.16	0.14	0.13
$(0.9, 0.8)$	0.1	1.17	0.83	0.65	0.54	0.48	0.43	0.38	0.35	0.90	0.58	0.43	0.34	0.29	0.26	0.22	0.19
	0.2	0.98	0.64	0.49	0.41	0.36	0.33	0.30	0.28	0.74	0.42	0.30	0.24	0.20	0.18	0.16	0.14
	0.3	0.88	0.54	0.41	0.34	0.31	0.29	0.26	0.24	0.65	0.34	0.24	0.19	0.16	0.15	0.13	0.12

Table 2: Values of the allocated service $\hat{t}(n)$ determined by Heuristic II for $n = 1(1)6, 8, 10$ for a range of problems with gamma $\Gamma(r, \nu)$ service times

dominated by the final linear terms. We have found that in all cases studied the throughput $\bar{\omega}([(\mu^*)^{-1}])$ which is to be found in (42) is considerably larger than the quantity $\bar{\omega}(\mu^*)$ found in (20). Hence a larger negative penalty is paid via the final term for increasing t in (42) than in (20). It is therefore not surprising that the (largest) maximum is to be found at a smaller value in (42) than in (20) and hence that $\hat{t}(n) \leq \bar{t}(n)$.

6 Numerical study

Tables 3(a) and 3(b) show estimates of the percentage of arriving tasks successfully served (namely, the throughput divided by the arrival rate) under five heuristics for a range of problems with gamma $\Gamma(r, \nu)$ service times. The problems studied are the same as those considered in Sections 4 and 5 and are such that there is a wide range (from under 5% to more than 80%) in the percentage of successfully served tasks. The five heuristics studied are (reading from left to right in the tables):

(i) the optimal Markovian policy μ^* discussed in Section 3;

(ii) Heuristic I as developed in Section 4;

(iii) the static policy $[(\mu^*)^{-1}]$ which allocates time $(\mu^*)^{-1}$ to all served tasks;

(iv) Heuristic II as developed in Section 5;

(v) a heuristic developed by the application of stochastic DP to a finite state/finite action/discrete time approximation of the problem, with a discrete time quantum set equal to $\delta = 0.002$.

In the case of (i) the figures quoted in the table were obtained by application of the formula in (4), while in (v) the methodology deployed was DP value iteration. See Tijms (1994). Please note that for the DP policy, a halving of the discrete time quantum to 0.001 did not improve the achieved throughput (to the accuracy reported in the tables) but greatly increased the computing time required. The results for (ii)–(iv) were obtained by Monte Carlo simulation. For the heuristics concerned, all estimates of the percentage of tasks successfully served are accompanied in the table by their standard errors (in brackets). The simulation study was designed to achieve sufficiently small standard errors to facilitate meaningful performance comparisons between the heuristics. To further assist the reader, the results for the optimal Markovian policy μ^* and for Heuristics I and II are displayed in Figures 1–3. The policy $[(\mu^*)^{-1}]$ and the DP heuristic were omitted from the figures because of their closeness in performance to Heuristics I and II respectively. See comments below.

Before preceding to discussion of the results, please note that (arbitrarily good approximations to) Heuristics I and II may be computed efficiently. We observe that under these (and, indeed, any other stationary) service heuristics, the number of tasks present in the system and *not* in receipt of service is stochastically bounded above in steady state by the number present in an equivalent system, but for which no service at all is offered. However, the steady state distribution of the latter is well known to be Poisson $(\lambda\theta^{-1})$. We infer that the long run proportion of time for which the queue length is outside the range $0 \leq n \leq \bar{N}$, where

$$\bar{N} = \max(50, \lambda\theta^{-1}) + 3\sqrt{\max(50, \lambda\theta^{-1})}$$

		$r = 2$				
(λ, ν)	θ	μ^*	HeurI	$[(\mu^*)^{-1}]$	HeurII	DP
	0.1	0.1554	0.2157	0.2182	0.2307	0.2271
			(0.0008)	(0.0009)	(0.0018)	
(0.25,0.3)	0.2	0.1050	0.1505	0.1508	0.1552	0.1560
			(0.0016)	(0.0016)	(0.0017)	
	0.3	0.0772	0.1118	0.1142	0.1148	0.1153
			(0.0013)	(0.0015)	(0.0015)	
	0.1	0.0617	0.0735	0.0678	0.0738	0.0753
			(0.0006)	(0.0006)	(0.0006)	
(0.9,0.3)	0.2	0.0469	0.0580	0.0564	0.0590	0.0594
			(0.0006)	(0.0006)	(0.0005)	
	0.3	0.0368	0.0466	0.0464	0.0472	0.0483
			(0.0005)	(0.0005)	(0.0005)	
	0.1	0.1775	0.2256	0.2279	0.2338	0.2337
			(0.0008)	(0.0010)	(0.0008)	
(0.9,0.8)	0.2	0.1467	0.1942	0.1964	0.2019	0.2024
			(0.0008)	(0.0009)	(0.0009)	
	0.3	0.1244	0.1687	0.1703	0.1748	0.1757
			(0.0008)	(0.0009)	(0.0009)	

		$r = 1$				
(λ, ν)	θ	μ^*	HeurI	$[(\mu^*)^{-1}]$	HeurII	DP
	0.1	0.3292	0.4493	0.4521	0.4582	0.4584
			(0.0025)	(0.0024)	(0.0028)	
(0.25,0.3)	0.2	0.2679	0.3656	0.3610	0.3703	0.3705
			(0.0024)	(0.0024)	(0.0026)	
	0.3	0.2298	0.3123	0.3128	0.3173	0.3163
			(0.0022)	(0.0025)	(0.0025)	
	0.1	0.1904	0.2309	0.2391	0.2432	0.2432
			(0.0009)	(0.0010)	(0.0010)	
(0.9,0.3)	0.2	0.1635	0.2055	0.2096	0.2139	0.2146
			(0.0009)	(0.0009)	(0.0009)	
	0.3	0.1448	0.1873	0.1891	0.1924	0.1930
			(0.0009)	(0.0010)	(0.0009)	
	0.1	0.3580	0.4691	0.4763	0.4879	0.4876
			(0.0012)	(0.0015)	(0.0013)	
(0.9,0.8)	0.2	0.3171	0.4254	0.4267	0.4344	0.4361
			(0.0011)	(0.0014)	(0.0012)	
	0.3	0.2887	0.3884	0.3911	0.3973	0.3982
			(0.0011)	(0.0013)	(0.0012)	

Table 3(a): Estimates of the percentage of arriving tasks successfully served under five heuristics for a range of problems with gamma $\Gamma(r, \nu)$ service times

		$r = 0.5$				
(λ, ν)	θ	μ^*	HeurI	$[(\mu^*)^{-1}]$	HeurII	DP
	0.1	0.5214	0.6530	0.6472	0.6556	0.6598
			(0.0032)	(0.0034)	(0.0035)	
(0.25,0.3)	0.2	0.4667	0.5823	0.5794	0.5842	0.5869
			(0.0034)	(0.0034)	(0.0034)	
	0.3	0.4315	0.5396	0.5363	0.5406	0.5403
			(0.0031)	(0.0033)	(0.0032)	
	0.1	0.3901	0.4848	0.4806	0.4924	0.4924
			(0.0014)	(0.0015)	(0.0015)	
(0.9,0.3)	0.2	0.3558	0.4429	0.4381	0.4469	0.4512
			(0.0013)	(0.0014)	(0.0014)	
	0.3	0.3331	0.4194	0.4142	0.4235	0.4228
			(0.0014)	(0.0015)	(0.0015)	
	0.1	0.5515	0.6912	0.6831	0.6960	0.6966
			(0.0016)	(0.0017)	(0.0017)	
(0.9,0.8)	0.2	0.5116	0.6441	0.6359	0.6474	0.6482
			(0.0017)	(0.0018)	(0.0017)	
	0.3	0.4846	0.6112	0.6037	0.6136	0.6139
			(0.0018)	(0.0019)	(0.0018)	

		$r = 0.25$				
(λ, ν)	θ	μ^*	HeurI	$[(\mu^*)^{-1}]$	HeurII	DP
	0.1	0.6859	0.7850	0.7792	0.7860	0.7951
			(0.0038)	(0.0038)	(0.0038)	
(0.25,0.3)	0.2	0.6460	0.7463	0.7421	0.7476	0.7463
			(0.0038)	(0.0038)	(0.0038)	
	0.3	0.6204	0.7123	0.7079	0.7120	0.7155
			(0.0039)	(0.0039)	(0.0039)	
	0.1	0.5909	0.6859	0.6765	0.6885	0.6904
			(0.0018)	(0.0019)	(0.0019)	
(0.9,0.3)	0.2	0.5601	0.6543	0.6448	0.6557	0.6555
			(0.0020)	(0.0021)	(0.0021)	
	0.3	0.5402	0.6297	0.6216	0.6299	0.6320
			(0.0019)	(0.0019)	(0.0019)	
	0.1	0.7111	0.8200	0.8108	0.8208	0.8242
			(0.0019)	(0.0020)	(0.0019)	
(0.9,0.8)	0.2	0.6800	0.7879	0.7783	0.7878	0.7891
			(0.0020)	(0.0020)	(0.0021)	
	0.3	0.6593	0.7644	0.7564	0.7647	0.7656
			(0.0019)	(0.0019)	(0.0019)	

Figı

Table 3(b): Estimates of the percentage of arriving tasks successfully served under five heuristics for a range of problems with gamma $\Gamma(r, \nu)$ service times

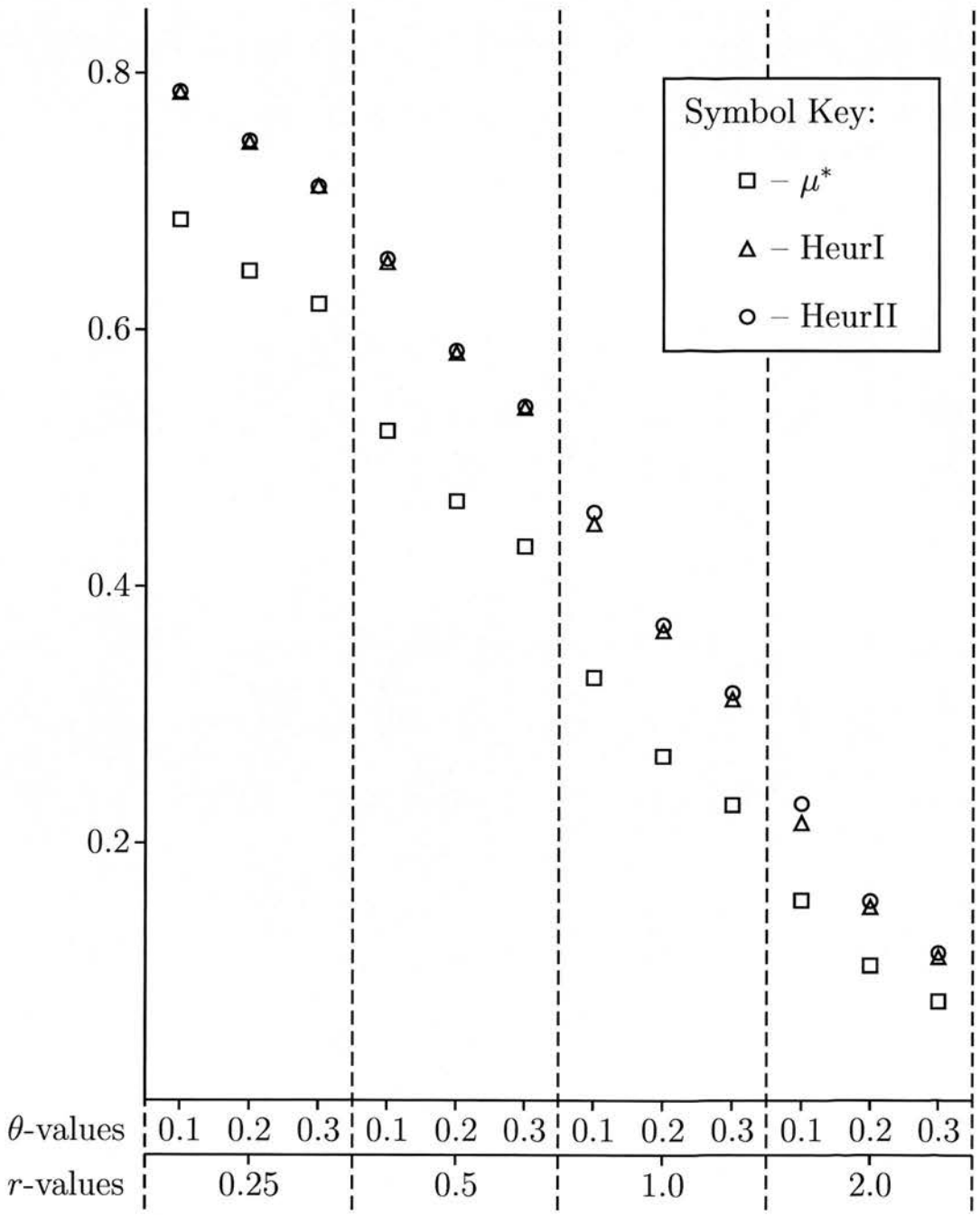

Figure 1: The percentage of arriving tasks successfully served under three heuristics for the case $(\lambda, \nu) = (0.25, 0.3)$

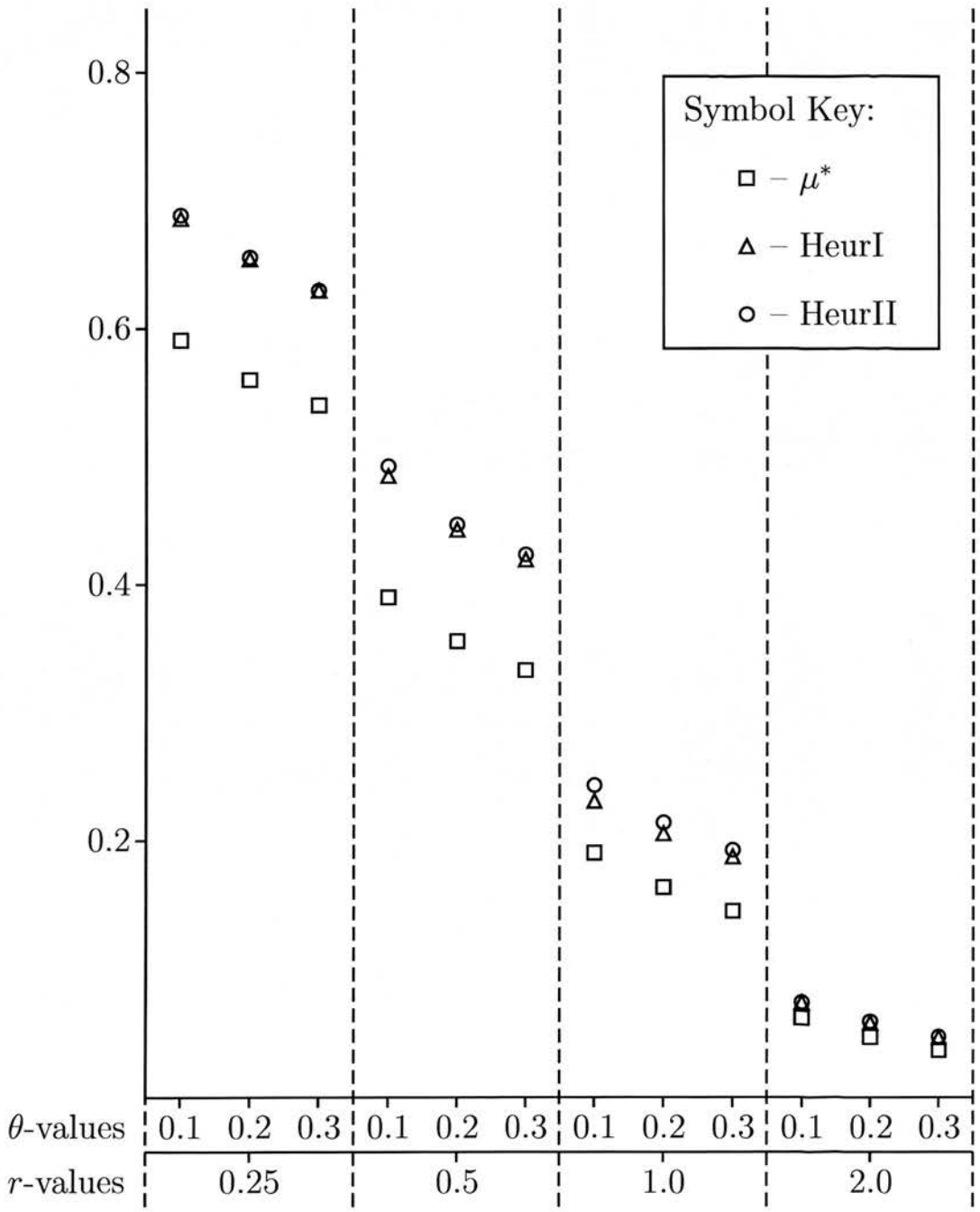

Figure 2: The percentage of arriving tasks successfully served under three heuristics for the case $(\lambda, \nu) = (0.9, 0.3)$

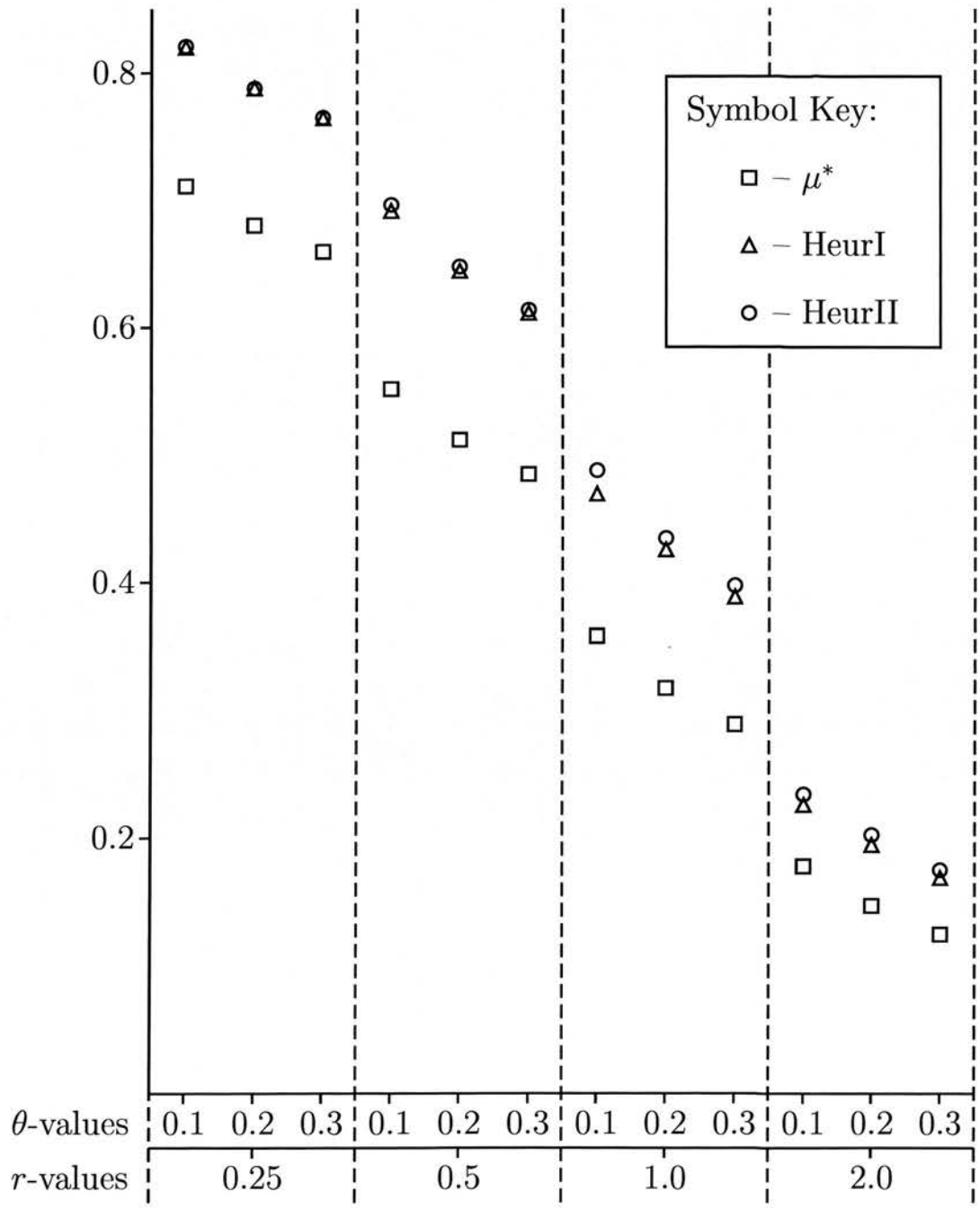

Figure 3: The percentage of arriving tasks successfully served under three heuristics for the case $(\lambda, \nu) = (0.9, 0.8)$

is small (at most around 10^{-3}). Hence implementation of the maximisations in (20) and (42) over the range $1 \leq n \leq \bar{N}$ is perfectly adequate. For values of n above $\bar{N}$ we set the values of $\bar{t}(n)$ and $\hat{t}(n)$ equal to $\bar{t}(\bar{N})$ and $\hat{t}(\bar{N})$ respectively. In practice, both $\bar{t}(n)$ and $\hat{t}(n)$ achieve limiting values before n reaches $\bar{N}$. The computational ease with which the dynamic heuristics I and II may be developed (in practice, this takes just a few seconds on a standard PC) stands in sharp contrast to the computational burden of a full DP implementation as outlined in Section 2.

The evidence of Tables 3(a) and 3(b) is that under all heuristics the percentage of successfully served tasks is decreasing in the exogenous arrival rate λ, the loss rate θ and the mean (actual) service time $r\nu^{-1}$. That for given (λ, ν), the percentage of successfully served tasks is decreasing in r and θ is particularly clear from Figures 1–3. The optimal Markovian policy μ^* performs poorly. An approach to the allocation of service times which is both static (i.e., state independent) and, more crucially, random with a high degree of variability does not work well. Overall, using the performance measures in Tables 3(a) and 3(b), Heuristic I effects an improvement of over 26% on the Markovian policy μ^*. This improvement is most marked in the low throughput/high mean service time cases. A notable feature of Tables 3(a) and 3(b) is the strong performance of the static policy $[(\mu^*)^{-1}]$, reflecting the fact that while the allocated services chosen by strongly performing dynamic policies do vary with the queue length, the degree of variability is modest. Overall, the performances of Heuristic I and the static policy $[(\mu^*)^{-1}]$ are similar, with the dynamic heuristic slightly outperforming the static policy in high throughput instances. Heuristic II outperforms all of the above. That it should outperform static policy $[(\mu^*)^{-1}]$ is guaranteed by Corollary 9. While the performances of Heuristics I and II are fairly comparable in high throughput instances, Heuristic II is plainly the stronger in cases (e.g., when $r = 2$) for which the percentage of successfully served tasks is low. Overall, Heuristic II offers an improvement in performance of around 2% on average over the static policy $[(\mu^*)^{-1}]$ but the degree of improvement is much greater than this in low throughput instances where it rises to 9%. In all cases studied, a comparison of the performance of Heuristic II with that of the policy developed by stochastic DP yields the conclusion that the former is likely very close to optimal. In the cases considered, DP offers an improvement in performance over Heuristic II of just 0.3% on average.

Acknowledgement

Both authors gratefully acknowledge support received from the Engineering and Physical Science Research Council. For the first author, the source of funding was the grant GR/S45188/01, while for the second it was through a research studentship. They also acknowledge the thoughtful comments of Professors Don Gaver and Pat Jacobs of the Naval Postgraduate School and of an associate editor and two referees which led to a range of improvements to the paper. Finally, the authors would like to thank Dr. Chris Kirkbride for his assistance in producing the figures.

References

Baccelli, F., Boyer, P. & Hebuterne, G. (1984), 'Single-server queues with impatient customers', *Adv. Appl. Prob.* **16**, 887–905.

Bassamboo, A., Harrison, J. M. & Zeevi, A. (2005), 'Dynamic routing and admission control in high-volume service systems: asymptotic analysis via multi-scale fluid limits', *Queueing Systems* **51**, 249–285.

Deshmukh, S. D. & Jain, S. (1977), 'Capacity design and service quality control in a queueing system', *Oper. Res.* **25**, 651–661.

Doytchinov, B., Lehoczky, J. P. & Shreve, S. (2001), 'Real-time queues in heavy traffic with earliest-deadline-first queue discipline', *Ann. Appl. Probab.* **11**, 332–378.

Garnett, O., Mandelbaum, A. & Reiman, M. (2002), 'Designing a call center with impatient customers', *Manuf. and Service Oper. Mgmt.* **4**, 208–227.

Gaver, D. P., Jacobs, P. A., Samorodnitsky, G. & Glazebrook, K. D. (2006), 'Modeling and analysis of uncertain time-critical tasking problems', *Nav. Res. Logist.* **53**, 588–599.

Gaver, D. P., Jacobs, P. A. & Sato, H. (2006), 'Assessing resource requirements for maritime domain awareness and protection (security)'. Working Paper, Department of Operations Research, Naval Postgraduate School, Monterey, CA.

Glazebrook, K. D. (1983), 'Stochastic scheduling with due dates', *Int. J. Syst. Sci.*, **14**, 1259–1271.

Glazebrook, K. D., Ansell, P. S., Dunn, R. T. & Lumley, R. R. (2004), 'On the optimal allocation of service to impatient tasks', *J. Appl. Prob.* **41**, 51–72.

Glazebrook, K. D. & Punton, E. L. (2005), 'Fixed-time schedules for the processing of jobs when service completions are not observable', *Math. Meth. Oper. Res.* **62**, 77–97.

Harrison, J. M. & Zeevi, A. (2004), 'Dynamic scheduling of a multi-class queue in the Halfin-Whitt Heavy Traffic Regime', *Oper. Res.* **52**, 243–257.

Jiang, Z., Lewis, T. G. & Colin, J. Y. (1996), 'Scheduling hard real-time constrained periodic tasks on multiple processors', *J. Syst. Software* **19**, 102–118.

Krishnan, K. R. (1987), 'Joining the right queue: a Markov decision rule', In *Proc. 28th IEEE Conf. Decision Control*, pp. 1863–1868.

Lehoczky, J. P. (1996), 'Real-time queueing theory', In *Proc. 17th IEEE Real-Time Systems Symp.*, December 1996, pp. 186–195.

Lehoczky, J. P. (1997a), 'Real-time queueing network theory', In *Proc. 18th IEEE Real-Time Systems Symp.*, December 1997, pp. 58–67.

Lehoczky, J. P. (1997b), 'Using real-time queueing theory to control lateness in real-time systems', *Perform. Eval. Rev.* **25**, 158–168.

Tijms, H. C. (1994), *Stochastic models: an algorithmic approach*, Wiley, Chichester.